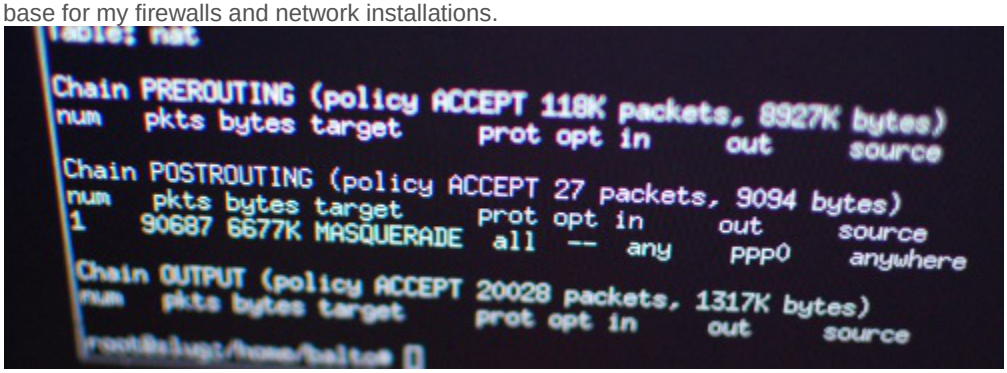
IPTABLES STATEFUL FIREWALL AND NAT ROUTING

Network packet filtering! Whether is your home or your company, modern networks have many systems connected. Even a small domestic network can provide connectivity to many devices of different kinds: PCs, laptops, printers, smartphones, game consoles, your neighbor's laptop (wait… what?!), NASes, media players, TVs…
If you have some basic knowledge in networking, you'll probably want a way to control all the traffic going through your network, and if you are running a GNU/Linux system, you probably already have what you need… Netfilter!
What you'll find here are some examples of common Netfilter (iptables) configurations and some scripts I use as a base for my firewalls and network installations.



These are really useful if you need some advanced firewall configuration and you choose to run your own GNU/Linux system as a router instead of a commercial one. Also, these scripts may come in handy if you need to quickly replace a broken router with a spare PC.

netfilter/iptables

First, the basics: *netfilter* is the framework used to manipulate packets in the network stack of modern Linux kernels and *iptables* is the userspace application used to write and control the chains of rules in the kernel.
Of course I'm not going to write an iptables HOWTO because the author already did. If you are interested you should read the official documentation, starting from the "Packet Filtering HOWTO".
Here I'll assume you have a basic knowledge of the iptables utility. The following paragraph will explain the main commands used in my firewall scripts.

Basic Configuration

Default policy is what the firewall does if the packet won't match any jump rule. I like to write rules that explicitly accept data and drop any unknown traffic, so let's just use a default INPUT DROP policy with:

```
iptables -P INPUT DROP
```

Also, a good idea is to accept all the packet on the loopback interface, as in

```
iptables -A INPUT -i lo -j ACCEPT
```

Finally, if you have a local network you trust, you can just accept all the packet from that, with

```
iptables -A INPUT -i eth0 -s 192.168.0.0/24 -j ACCEPT
```

Stateful Filtering

What most people need in term of network security, is to drop any unexpected incoming traffic and accept all the traffic initiated by the host.

This means that the filter have to keep track of all the connections, even the stateless ones, which in firewall language is called Stateful Packet Inspection. Then we can configure a rule to accept only the traffic for which we have some information because the stream was accepted by some other rule, like the default OUTPUT policy.
The rule to do that, using the *conntrack* module is:

```
iptables -A INPUT -i ppp0 -m state --state ESTABLISHED,RELATED \

         -j ACCEPT
```

which means: accept any traffic coming into ppp0 which is part of or can be associated with an existing connection. This kind of protection should be enough for a networked host which doesn't want to expose any service on non authorized networks.

NAT/IP Masquerading

In the time when routers where very expensive and people started feel the need to share a domestic internet connection with all their systems (does anyone remember WinGate?), IP masquerading was considered a killer feature of Linux network stack, so there are *many* tutorials around on how to do that.
The thing by itself is pretty easy: you have two (or more) network interfaces and you want to route traffic from one to another modifying it as it always appears to be originated by the routing host. That's commonly known as NAT.
An example where our local network is on eth0 and the remote is on ppp0:

```
balto@slug:~$ /sbin/ifconfig

eth0      Link encap:Ethernet  HWaddr 00:13:10:D7:15:AA

          inet addr:192.168.0.1  Bcast:192.168.0.255

          Mask:255.255.255.0

...

ppp0      Link encap:Point-to-Point Protocol

          inet addr:87.4.175.15  P-t-P:192.168.100.1

          Mask:255.255.255.255
```

First, enable routing of ipv4 packets:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

then, configure the kernel to use IP masquerading on the output interface for traffic coming from the other one:

```
iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o ppp0 \

         -j MASQUERADE
```

That's about it! If you want to see the actual connection tracking table, you can dump it by using:

```
root@slug:~# cat /proc/net/nf_conntrack

...

ipv4 2 tcp 6 299 ESTABLISHED src=192.168.0.3 dst=74.200.247.59

  sport=34880 dport=443 packets=3 bytes=368 src=74.200.247.59

  dst=87.4.175.15 sport=443 dport=34880 [ASSURED] mark=0 use=1

...
```

Where you can observe that the translated connections have different IPs for send and return path. Here 192.168.0.3 is my netbook, 74.200.247.59 is a wordpress.com webserver and 87.4.175.15 is my router's public IP.

Reverse NAT (DNAT)

Once you have configured your network to use IP masquerading, the devices inside your network will be able to communicate with the outside world, but because the routing host is translating the addresses, the external system will not have direct visibility of the internal one.

That's usually good because it intrinsically gives you the security of a stateful firewall.

If for some reason you want to expose an internal service to the outside world, you just need to add a "DNAT" rule to map that service on a port of the routing host, such as in:

```
iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 80 \

         -j DNAT --to 192.168.0.3:80
```

This example will make the http service of the host 192.168.0.3 available on the http port of the ppp0 interface of the router.

That kind of service-mapping was pretty common with older peer-to-peer applications. Nowadays, they usually employ some NAT-traversal technique, so that they don't need that kind of reverse NAT mappings anymore.

NAT on PPPoE connections (aka: the MTU problem)

When routing packets between network interfaces, you should pay attention that the two interfaces might have different MTU configurations.

That's very common for domestic ADSL connections, as the PPPoE protocol takes 8 bytes out of every Ethernet packet, thus reducing the MTU of the PPP interface to 1492 bytes instead of the usual 1500.

Check out the *ifconfig* output:

```
eth0      Link encap:Ethernet  HWaddr 00:13:10:D7:15:AA

...

          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1



ppp0      Link encap:Point-to-Point Protocol
```

```
...

          UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1492 Metric:1
```

This kind of configuration is not a problem for the actual host, as part of the TCP handshake protocol implies the exchange of MSS information, which is derived from the interface's MTU.
On the other side, if you are using that node as a NAT router, the systems behind it have no way to know the real MTU of the PPPoE interface. Therefore the systems will try to use packets bigger than the maximum allowed, which will be dropped without warning by routers.
The solution for that, unless you want to configure all your devices with a reduced MTU, is to instruct the routing host to intercept all the TCP handshake packets and correct in-fly the wrong MSS value requested by internal hosts.
The rule to do that is the following:

```
iptables -t mangle -A FORWARD -p tcp --tcp-flags SYN,RST SYN

         -j TCPMSS --clamp-mss-to-pmtu
```

rc.firewall scripts

The firewall script should be placed with other system initialization scripts and called automatically during the startup of the system.
I call my firewall scripts rc.firewall, as this is the name used by Slackware scripts. I have a couple of them I use quite often on my systems:
rc.firewall: a firewall script for router hosts - it uses eth0 as internal network and ppp0 as external by default.
rc.firewall.standalone: a local stateful firewall for mobile hosts - it trust the wired network and filter wireless one by default.

Hardware

If you like controlling your own firewall, you'll probably want to use some embedded device with a router-like form factor to run your favorite GNU/Linux distribution on it. Just search for some passively cooled device for noiseless operation... I know of Soekris and PC Engines ones, but some random Atom Mini-ITX with a HTPC case will do the job.
Personally, I use a Linksys NSLU2 (Intel Xscale) with my custom 5 port VLAN switch, so that the switch ports can be seen as separate interfaces by the system.

Source : http://fabiobaltieri.com/2011/09/12/iptables-firewall-nat/