Building a Home Gateway/Firewall with Linux

(aka “Firewalling and NAT with iptables”)

Michael “Porkchop” Kaegler
mkaegler@nic.com
http://www.nic.com/~mkaegler/

Hardware Requirements

- Any machine capable of running linux
- Network interfaces - at least one internal and one external.

Software Requirements

- v2.4 kernel
  - Networking Options -> TCP/IP Networking -> Network Packet Filtering
  - Networking Options -> TCP/IP Networking -> IP: advanced router -> (everything)
  - Networking Options -> IP: Netfilter Configuration -> (everything as modules)
  - *Optional for traffic shaping:* Networking Options -> QoS and/or fair queueing -> (everything)

A quick word on addressing

- 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 are "nonroutable" and can be used for your internal networks.
  - Address Range: 192.168.1.1 thru 192.168.1.254
  - Subnet Mask: 255.255.255.0 (aka "class c" or "/24")
  - Network Address: 192.168.1.0
  - Broadcast Address: 192.168.1.255

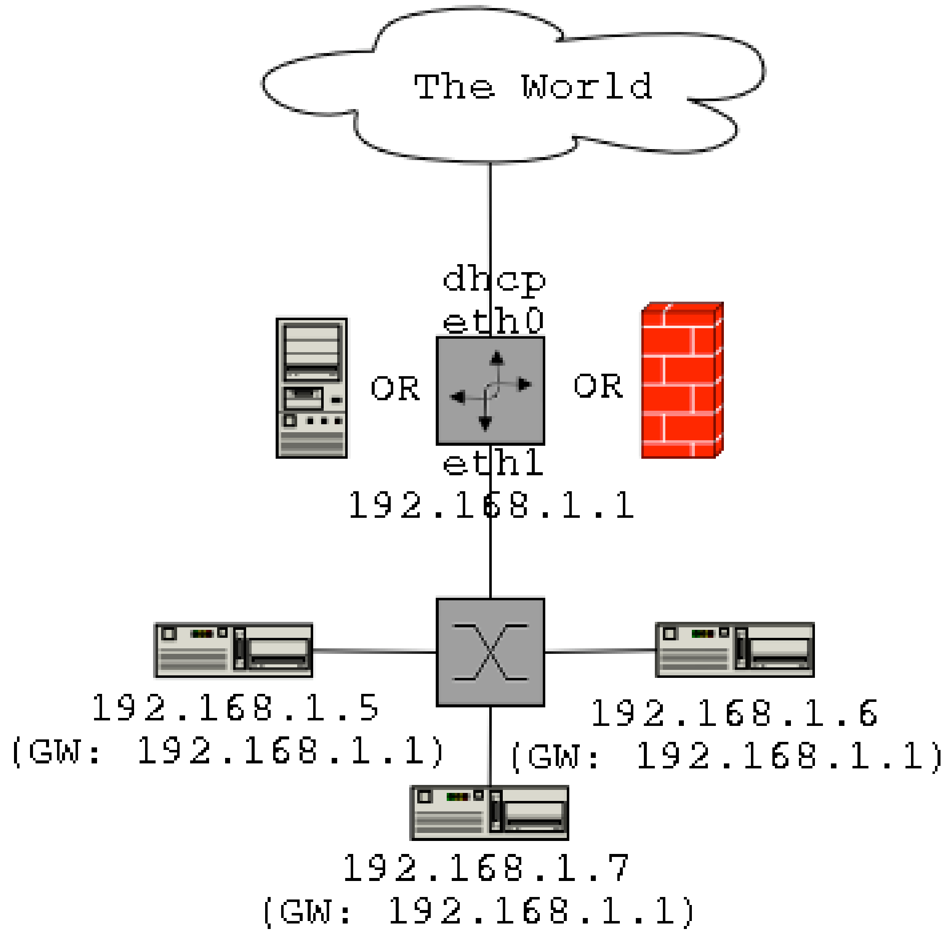
The internal network

Address Range: 192.168.1.1 thru .254

Subnet Mask: 255.255.255.0

Network Address: 192.168.1.0

Broadcast Address: 192.168.1.255



Make sure the machine can talk to ALL the networks it will need to.
To enable straight routing:

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

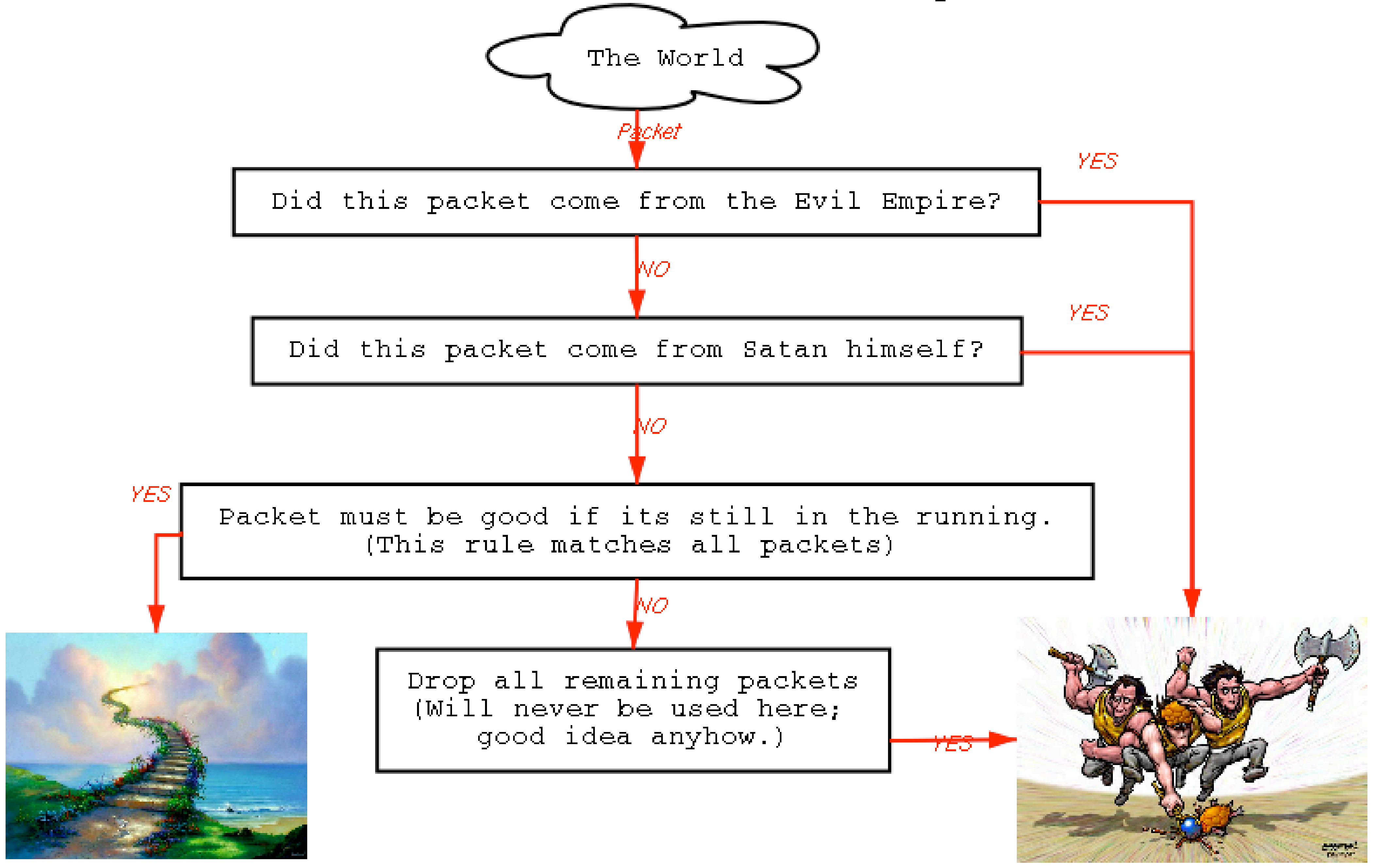
Why firewall?

- Breaking into systems is done via exploiting software bugs.
  - Limiting attackers ability to talk to your software limits their ability to exploit your softwares' bugs.
- Tracking/Stats

Why NAT?

- If you only have one IP address
- Simplicity?

Basic function of iptables


The World
Packet
Did this packet come from the Evil Empire?
YES
NO
Did this packet come from Satan himself?
YES
NO
YES
Packet must be good if its still in the running.
(This rule matches all packets)
NO
Drop all remaining packets
(Will never be used here;
good idea anyhow.)
YES

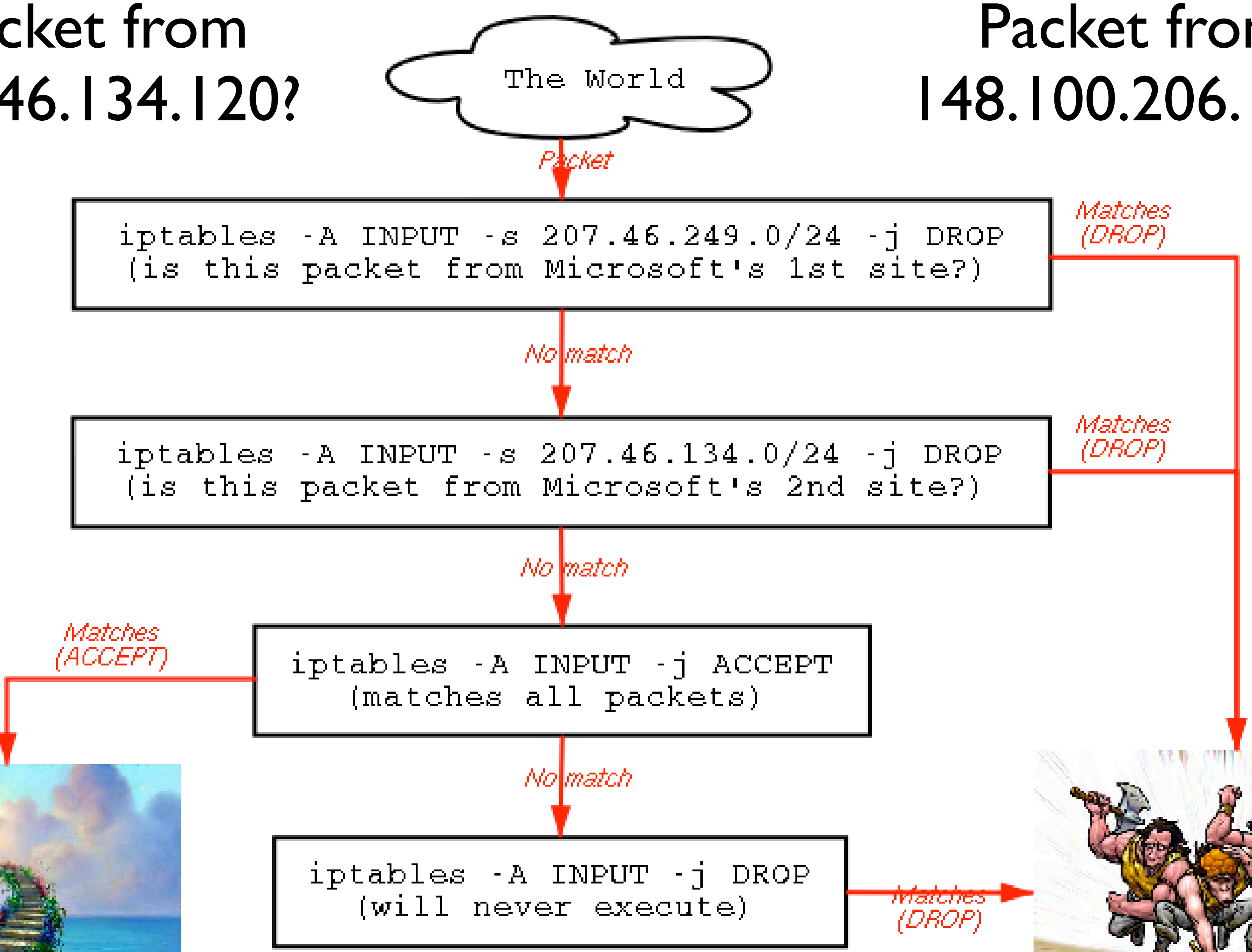
FW: How packets get in and out

- “filter” (default) table: 3 chains (think ‘paths’ or ‘flows’):
  - INPUT for packets destined for the host itself.
  - OUTPUT for packets sent by the host itself.
  - FORWARD for packets just being routed (includes NAT)
- 2 basic targets
  - ACCEPT allows a packet.
  - DENY drops it.

FW: Looking at chains

Packet from 207.46.134.120?

Packet from 148.100.206.150?

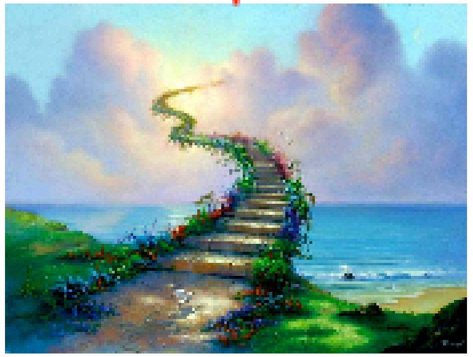


```
iptables -A INPUT -s 207.46.249.0/24 -j DROP
(is this packet from Microsoft's 1st site?)

iptables -A INPUT -s 207.46.134.0/24 -j DROP
(is this packet from Microsoft's 2nd site?)

iptables -A INPUT -j ACCEPT
(matches all packets)

iptables -A INPUT -j DROP
(will never execute)
```



FW: Forming your own rules

Basic firewalling rule:

iptables -A source-chain (match spec) -j target

-A source-chain: "Add this statement to the end of this chain"

(match spec): "If all these rules are satisified, the packet matches"

-j target: "If the packet matches, jump it to here (other chains are valid targets)"

Match Specs: There are scads. Check man pages for a full list; these are the big players.

-p, --protocol protocol

    tcp/udp: --source-port [port[:port]], --destination-port [port[:port]]

    icmp: --icmp-type typename

-s, --source address[/mask]

-d, --destination address[/mask]

-i, --in-interface [name]

-o, --out-interface [name]

-f, --fragment (second+ only)

Important Modifier: ! (negates value)

"-p tcp" matches all tcp packets. "-p ! tcp" matches all packets that are not tcp.

FW: Relevent Modules

```
modprobe ip_tables      # the whole thing

#Match extentions:
modprobe ipt_limit      # Match against rate of occurence. Think log extention.
modprobe ipt_mac        # Match MAC addresses.
modprobe ipt_mark       # Match/mark packets. useful w/ traffic shaping.
modprobe ipt_multiport  # multiport...dunno why this needs a special module.
#modprobe ipt_tos       # Match Type Of Service
#modprobe ipt_esp       # Match against SPIs in ESP headers...IPSec stuff.
#modprobe ipt_ah        # Match against SPIs in AH headers...IPSec stuff.
#modprobe ipt_length    # Match packet length. My packet is bigger...nevermind.
#modprobe ipt_ttl       # Match against TTL. Could be used to mess with wintel.
#modprobe ipt_tcpmss    # Match tcpmss (a field in SYN packets that set maxsize)
modprobe ipt_state      # Match the current connection state.
#modprobe ipt_unclean   # Match non-kosher packets. Experimental at best.
#modprobe ipt_owner     # Match by user, group, process, or session owner. Cool.

#Tables and their targets.
modprobe iptable_filter # Table. INPUT, OUTPUT, and FORWARD chains.
 modprobe ipt_REJECT    #  Target. Like DROP but makes ICMP errors.
# modprobe ipt_MIRROR   #  Target. The sender. Imagine portscans. Experimental.
modprobe iptable_mangle # Table. To mangle packets.
 modprobe ipt_TOS       #  Target. Alter the TOS field prior to routing.
 modprobe ipt_MARK      #  Target. Alter the mark of a packet.
modprobe ipt_LOG        # Target. Syslog.
#modprobe ipt_ULOG      # Target. Userspace logger. gnumonks.org/projects/ulogd
#modprobe ipt_TCPMSS    # Target. Alter the TCPMSS.
# NAT table is later
```

FW: Some rules

```
###
### FIREWALL FOR THIS MACHINE
###
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i eth4 -p tcp --sport 67 --dport 68 -j ACCEPT
iptables -A INPUT -i ! eth4 -p tcp --sport 68 --dport 67 -j ACCEPT
iptables -A INPUT -i eth4 -p udp --sport 67 --dport 68 -j ACCEPT
iptables -A INPUT -i ! eth4 -p udp --sport 68 --dport 67 -j ACCEPT
# that's all for dhcp.
iptables -A INPUT -i ! eth4 -j Internal
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 20 -j ACCEPT
iptables -A INPUT -p tcp --dport 21 -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 113 -j ACCEPT
iptables -A INPUT -p tcp --dport 1024: -j ACCEPT
iptables -A INPUT -p udp --dport 1024: -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
iptables -A INPUT -j DROP
```

FW: Rules for the FORWARD chain

```
# make new chains
iptables -N ROutbound
iptables -N RInbound
iptables -N LogServer
iptables -N Internal
# note these already exist: INPUT, OUTPUT, FORWARD

### FIREWALL FOR OTHER MACHINES
iptables -A FORWARD -i ! eth4 -o ! eth4 -j Internal
iptables -A Internal -s 216.169.91.88/29 -j ACCEPT
iptables -A Internal -i eth0 -j ACCEPT
iptables -A Internal -i eth1 -s 10.10.10.0/24 -j ACCEPT
iptables -A Internal -i eth2 -s 10.10.50.0/24 -j ACCEPT
iptables -A Internal -i eth3 -s 10.10.12.0/24 -j ACCEPT
iptables -A Internal -i wlan0 -s 10.10.11.0/24 -j ACCEPT
iptables -A Internal -i ppp0 -s 10.10.13.0/24 -j ACCEPT
iptables -A Internal -j DROP
# While we're inside the building, we can say
# whatever we want.
# Just be sure you could be who you claim to be.
```

FW: Rules for the FORWARD chain

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# Packets from established connections are groovy.

iptables -A FORWARD -d 216.169.91.90 -j ACCEPT
iptables -A FORWARD -d 216.169.91.91 -j ACCEPT
# These two machines firewalled deeper in the network.

iptables -A FORWARD -i eth4 -o ! eth4 -j RInbound
iptables -A FORWARD -i ! eth4 -o eth4 -j ROutbound
# RInbound now contains traffic from outside for the inside.
# ROutbound now contains traffic from inside for the outside.

iptables -A RInbound -d ! 216.169.91.88/29 -j DROP
iptables -A RInbound -s 216.169.91.88/29 -j DROP
# we only own 216.169.91.88/29, so packets coming in on
# eth4 for other stuff,
# or claiming to be from possibly trusted hosts...
```

FW: Rules for the FORWARD chain

```
iptables -A ROutbound -d ! 216.169.91.88/29 -j ACCEPT
# Nothing not destined for a host inside should be headed out.
# Source hosts: 216.169.91.88/29, 10.0.0.0/8

iptables -A RInbound -p tcp --dport 22 -j ACCEPT
iptables -A RInbound -p udp --dport 33434 -j ACCEPT
#Everyone loves ssh! Oh, and traceroute udp packets too.

iptables -A RInbound -d 216.169.91.92 -j LogServer
iptables -A LogServer -p udp --dport 514 -s 126.74.85.184 -j ACCEPT
iptables -A LogServer -p udp --dport 514 -s 148.100.0.0/16 -j ACCEPT
#We only want to let some hosts in...in case of some stupid buffer
overflow.

iptables -A RInbound -j DROP
iptables -A ROutbound -j DROP
#Close up the loose ends.

echo 1 > /proc/sys/net/ipv4/ip_forward
#Enableing forwarding
```

Adding NAT

modprobe the modules to add the NAT table to ipchains, as well as its associated targets and helpers:

```
modprobe iptable_nat        # Table. Network Address Translation
 modprobe ipt_MASQUERADE    # Target. Self explanitory.
  modprobe ip_conntrack     # connection tracking; used for nat
  modprobe ip_conntrack_ftp # special connection tracking for ftp...
  modprobe ip_conntrack_irc # ...and irc
```

Add rule to the POSTROUTING chain (part of the NAT table): stuff that wants to be routed out eth0 and is from an address on the internal network gets MASQUERADEd:

```
iptables -A POSTROUTING -t nat -o eth0 -s 192.168.1.0/24 -j MASQUERADE
```

Other Good Ideas(tm) for firewall config scripts

```
/usr/local/bin/snort -c /etc/snort.conf -i eth4 -q &
# http://www.snort.org/

echo 1 > /proc/sys/net/ipv4/tcp_syncookies;
# TCP SynCookie support. In menuconfig,
# Networking options -> IP: TCP syncookie support
# Disabled by default because clients may not display errors when
#  server is under heavy load.

if [ -f "/proc/sys/net/ipv4/conf/all/rp_filter" ]; then
    for x in `ls /proc/sys/net/ipv4/conf/*/rp_filter`; do
    echo 1 > $x;
    done
fi
# rpfilter (part of forwarding) protects against spoofing by being
#  sure that incoming traffic could have come in on the interface it
#  did.

echo 1 > /proc/sys/net/ipv4/ip_forward
# Enabling forwarding (routing)
```