Взято: http://www.gentoo-wiki.info/HOWTO_Iptables_for_newbies

Iptables is a userspace command line program that can plug into netfilter (a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack). Iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel.

## Contents

# Introduction

I found the iptables documentation available to be severely unfriendly to newbies. Most assumed a more than working knowledge of ipchains and pretty much picked up from there. Usually my approach to a new endeavor is that I want the option of getting up and going quickly, with minimal explanation. Afterwards I go back to read over more advanced options. This howto is designed with that in mind. It will show you how to setup basic functionality get you connected quickly and then it will incrementally show how to add policies and rules that can help a safe connection.

# Installing iptables

## Enable Kernel Support

You'll need to compile iptables support into the kernel (either within the kernel itself or through modules). You can use either a manual configuration or sys-kernel/genkernel . Either way, go to your kernel sources and modify your configuration:

```
cd /usr/src/linux
make menuconfig
```

Once you get to the menu configuration you will need to enable to following depending on your kernel version and whether or not you want to also support IPv6. If you're just a newbie, then just go ahead and enable all of the options as shown below because it will be the quick and dirty way of getting everything working at the expense of increase your kernel's size.

```
        Linux Kernel Configuration: Netfilter for kernel version 2.6.20
      Networking ---->
 Networking options ---->
  [*] Network packet filtering framework (Netfilter)--->
   Core Netfilter Configuration ---->
     Netfilter connection tracking support
     Netfilter Xtables support (required for ip_tables)
     "state" match support
   IP: Netfilter Configuration --->
     IPv4 connection tracking support (required for NAT)
     IP tables support (required for filtering/masq/NAT)
       Packet Filtering
         REJECT target support

             Linux Kernel Configuration: Netfilter for kernel version 2.6.22
      Networking ---->
 Networking options ---->
  Network packet filtering framework (Netfilter)--->
   Core Netfilter Configuration ---->
     Netfilter connection tracking support
     Netfilter Xtables support (required for ip_tables)
       "NFLOG" target support
       "conntrack" connection tracking match support
       "state" match support
   IP: Netfilter Configuration --->
     IPv4 connection tracking support (required for NAT)
     IP tables support (required for filtering/masq/NAT)
       Packet Filtering
         REJECT target support
       Full NAT
         MASQUERADE target support
```

When compiling 2.6.26-r1, be sure to add IP mangling support:

```
        Linux Kernel Configuration: Netfilter for kernel version 2.6.26-r1
      Networking ---->
 Networking options ---->
```

```
Network packet filtering framework (Netfilter)--->
 IP: Netfilter Configuration --->
  Packet mangling
```

If you intend to load iptables as a module (meaning you selected **M** instead of * for various kernel options), don't forget to enable automatic kernel module loading:

```
        Linux Kernel Configuration: Loadable Module Support
      [*] Enable loadable module support --->
[*]   Automatic kernel module loading
```

**Note:** You can also enable options as modules, but don't forget to load them when you want to use them via modprobe or autoloading with /etc/modules.autoload.d/kernel-2.6 .

Now, compile and install your kernel.

### Emerging iptables

The kernel components you added to your kernel will not do anything without the iptables package installed. To install it, emerge net-firewall/iptables .

```
# emerge iptables
```

## Configuring iptables

**Note:** Before you begin configuring, if you built any of the kernel components m

```
# modprobe iptable_filter
```

### Rules

Iptables moves packets based on a list of rules composed of two components: a match and an effect. For every network packet that iptables inspects, it checks the list and picks the first rule that the packet satisfies. If there is no match, nothing special happens. If there's a match, the effect described for that match is executed on the packet.

**Warning:** Since iptables selectes the first rule that matches, the order of your rules does matter!

## Basic Configuration

The average home user might want something that accepts certain incoming connections but blocks everything else. Open /etc/iptables.bak in your favorite text editor and insert the following text.

**File:** /etc/iptables.bak

```
# Generated by iptables-save v1.2.11 on Tue May 10 08:06:58 2005
*filter
:INPUT ACCEPT [5:952]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [1192099:595387635]

# accept all from localhost
-A INPUT -s 127.0.0.1 -j ACCEPT

# accept all previously established connections
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# permit people to ssh into this computer
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT

# permit ftp and web hosting services
-A INPUT -p tcp -m state --state NEW -m tcp --dport 20 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 21 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT

# permit windows file sharing
-A INPUT -p tcp -m state --state NEW -m tcp --dport 137:139 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 426 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 445 -j ACCEPT

# permit five ports for bitorrent
-A INPUT -p tcp -m state --state NEW -m tcp --dport 6881:6886 -j ACCEPT
```

```
# reject all other packets coming into the computer, even from other computers in the local area network
-A INPUT -j REJECT --reject-with icmp-port-unreachable

COMMIT
# Completed on Tue May 10 08:06:58 2005
```

You can now load the above file using the following command: Now, you can tell iptables about this file:

```
# iptables-restore /etc/iptables.bak
```

Although the rules have now been added to iptables, they are not saved explicitly. As a result, you should have iptables save the newly added rules using the following command:

```
# /etc/init.d/iptables save
```

**Note:** If you want to make a backup of the rules you have added, you can choose /var/lib/iptables/rules-save instead of using the default.

Now try and start iptables:

```
# /etc/init.d/iptables start
# /etc/init.d/iptables stop
# /etc/init.d/iptables start
```

The reason we start, then stop, then start again is because we haven't yet started the iptables script, so we must set the "initialized" status before stopping. Stopping essentially erases all settings and puts you back to zero.

If everything works well and you don't seem to have lost access to the server, then you can add iptables to the default runlevel. This will have the iptables service start up each time you boot the computer.

```
# rc-update add iptables default
```

## GUI Alternatives

There are three programs that are designed to allow you to configure iptables easily:

- net-firewall/fwbuilder
- net-firewall/kmyfirewall (for KDE users)
- net-firewall/guarddog (uses QT interface)

## Firewall Hardening (Advanced Configuration)

In the following, we're going to further secure our now functional firewall. At the end of this section, you should now have a set of tested rules and policies that will prevent not only attacks to your own computer, but also attacks from your computer to the Internet. Protecting others from the possibility of being attacked by one of our compromised computers is an essential--and often--overlooked aspect of security. It is also common Internet courtesy and probably a very important aspect for a small office/home office (SOHO) network. Normally, virus infection is only a minor nuisance to a small network and rarely results in data loss. However, since small SOHO networks are often less secure than larger corporate networks, they are a favorite target for crackers looking for a launchpad" for denial of service (DoS) attacks or other underhanded skullduggary.

**Note:** The following is offered in a piecemeal fashion in a sequence that enable

### Setting up environment variables

We will define our networks interfaces and various tools used in the script:

**File:** newfirewall.sh

```
#!/bin/sh
#
# ********** VARIABLE DEFINITIONS **********
#
# External interface
EXTIF="eth0"
# Internal interface
INTIF="eth1"
# Loop device/localhost
```

```
LPDIF="lo"
LPDIP="127.0.0.1"
LPDMSK="255.0.0.0"
LPDNET="$LPDIP/$LPDMSK"
# Text tools variables
IPT="/sbin/iptables"
IFC="/sbin/ifconfig"
G="/bin/grep"
SED="/bin/sed"
AWK="/usr/bin/awk"
ECHO="/bin/echo"
# Setting up external interface environment variables
# Set LC_ALL to "en" to avoid problems when awk-ing the IPs etc.
export LC_ALL="en"
EXTIP="`$IFC $EXTIF|$AWK /$EXTIF/'{next}//{split($0,a,":");split(a[2],a," ");print a[1];exit}'`"
EXTBC="255.255.255.255"
EXTMSK="`$IFC $EXTIF|$G Mask:|$SED 's/.*Mask:\([^ ]*\)/\1/'`"
EXTMSK="`$IFC $EXTIF|$AWK /$EXTIF/'{next}//{split($0,a,":");split(a[4],a," ");print a[1];exit}'`"
EXTNET="$EXTIP/$EXTMSK"
$ECHO "EXTIP=$EXTIP EXTBC=$EXTBC EXTMSK=$EXTMSK EXTNET=$EXTNET"
# Due to absence of EXTBC I manually set it to 255.255.255.255
# this (hopefully) will serve the same purpose
# Setting up environment variables for internal interface
INTIP="`$IFC $INTIF|$AWK /$INTIF/'{next}//{split($0,a,":");split(a[2],a," ");print a[1];exit}'`"
INTBC="`$IFC $INTIF|$AWK /$INTIF/'{next}//{split($0,a,":");split(a[3],a," ");print a[1];exit}'`"
INTMSK="`$IFC $INTIF|$AWK /$INTIF/'{next}//{split($0,a,":");split(a[4],a," ");print a[1];exit}'`"
INTNET="$INTIP/$INTMSK"
$ECHO "INTIP=$INTIP INTBC=$INTBC INTMSK=$INTMSK INTNET=$INTNET"
```

## Working with ACCEPTs

This first part will have you work with ACCEPTs. ACCEPTs allows other computers to communicate with our server. In reality, this is very ill advised. A solid firewall policy should really DENY, then ACCEPT. But DENYing now will cause you to lose all connections while you are testing - a very bad thing to happen when you are not sure if your ACCEPT rules work at all. Therefore, while we are entering this first, it will be the second to last rule set in the final script.

**File:** accepts-firewall.sh

```
$IPT -t nat -A PREROUTING -j ACCEPT
# $IPT -t nat -A POSTROUTING -o $EXTIF -s $INTNET -j SNAT --to $EXTIP
```

```
# Comment out next line (that has "MASQUERADE") to not NAT internal network
$IPT -t nat -A POSTROUTING -o $EXTIF -s $INTNET1 -j MASQUERADE
$IPT -t nat -A POSTROUTING -o $EXTIF -s $INTNET2 -j MASQUERADE
$IPT -t nat -A POSTROUTING                    -j ACCEPT
$IPT -t nat -A OUTPUT                      -j ACCEPT
$IPT -A INPUT   -p tcp --dport auth --syn -m state --state NEW -j ACCEPT
$IPT -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

## iptables DROP & REJECT

Next we are going to define a couple of custom chains which will log DROP and REJECT events. This way we don't have to enter a separate line for each command entered. The logs will be sent to where your syslog default log messages are sent (usually /var/log/messages). Later I'm going to write a grep/sed script that will parse and organize these for easy viewing and set it as a daily cron job. This should be inserted immediately after the above definitions. When you are done, run the script again. It should have no affect on functionality of the network since we're just setting definitions. But it will ensure that we have no errors thusfar.

**File:** logging-firewall.sh

```
# ********** LOGGING CHAINS **********
#
# We are now going to create a few custom chains that will result in
# logging of dropped packets. This will enable us to avoid having to
# enter a log command prior to every drop we wish to log. The
# first will be first log drops the other will log rejects.
# Do not complain if chain already exists (so restart is clean)
$IPT -N DROPI   2> /dev/null
$IPT -A DROPI -m limit --limit 3/minute --limit-burst 10 -j LOG --log-prefix 'FIREWALL DROP BLOCKED
$IPT -A DROPI   -j DROP
$IPT -N REJECTI 2> /dev/null
$IPT -A REJECTI -m limit --limit 3/minute --limit-burst 10 -j LOG --log-prefix 'FIREWALL REJECT BLOCI
$IPT -A REJECTI -j REJECT
$IPT -N DROP2   2> /dev/null
$IPT -A DROP2 -m limit --limit 3/second --limit-burst 10 -j LOG --log-prefix 'FIREWALL DROP UNKNOW
$IPT -A DROP2   -j DROP
$IPT -N REJECT2 2> /dev/null
$IPT -A REJECT2 -m limit --limit 3/second --limit-burst 10 -j LOG --log-prefix 'FIREWALL REJECT UNKI
$IPT -A REJECT2 -j REJECT
# For testing, a logging ACCEPT chain
```

```
$IPT -N ACCEPTl   2> /dev/null
$IPT -A ACCEPTl -m limit --limit 10/second --limit-burst 50 -j LOG --log-prefix 'FIREWALL ACCEPT:'
$IPT -A ACCEPTl   -j ACCEPT
```

## iptables flushing

Ok, now that we see our devices are being detected properly, we are going to insert a flush commands. So that when our rules are assigned it will be done cleanly. These lines should be inserted after our utilities definitions, the last one being:

**File:** sed-script.sed

```
SED='/bin/sed'
# Flush all existing chains and erase personal chains
 CHAINS=`cat /proc/net/ip_tables_names 2>/dev/null`
 for i in $CHAINS
 do
  $IPT -t $i -F
 done
 for i in $CHAINS
 do
  $IPT -t $i -X
 done
```

## Local interfaces

Now we're ready to start laying down some rules. First we are going to accept all packets from our loopback device if the ip address matches that of any of our local interfaces:

```
  $IPT -A INPUT   -i $LPDIF -s   $LPDIP   -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $EXTIP   -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $INTIP1  -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $INTIP2  -j ACCEPT
```

## Blocking broadcasting

Now we will block broadcasts both incoming and outgoing. This can prevent DoS attacks against us, as well as preventing our clients from being used to DoS someone else. This is part of what's called "Egress Protection". It's a do unto your neighbour sort of philosophy. If all SysAdmins followed this policy, than many of the more severe and costly DoS attacks would either not have occurred or been extremely limited.

```
 # Blocking Broadcasts
$IPT -A INPUT   -i $EXTIF  -d  $EXTBC   -j DROPl
$IPT -A INPUT   -i $INTIF1 -d  $INTBC1  -j DROPl
$IPT -A INPUT   -i $INTIF2 -d  $INTBC2  -j DROPl
$IPT -A OUTPUT  -o $EXTIF  -d  $EXTBC   -j DROPl
$IPT -A OUTPUT  -o $INTIF1 -d  $INTBC1  -j DROPl
$IPT -A OUTPUT  -o $INTIF2 -d  $INTBC2  -j DROPl
$IPT -A FORWARD -o $EXTIF  -d  $EXTBC   -j DROPl
$IPT -A FORWARD -o $INTIF1 -d  $INTBC1  -j DROPl
$IPT -A FORWARD -o $INTIF2 -d  $INTBC2  -j DROPl
```

Now test the script once more to ensure we have no syntax errors. Also notice that we are using our newly defined DROPl chain. This means that the dropped packets will be logged.

## Block WAN to LAN

Next we are going to block WAN access to our LAN if not specifically intended for our ISP assign IP:

```
 # Block WAN access to internal network
# This also stops nefarious crackers from using our network as a
# launching point to attack other people
# iptables translation:
# "if input going into our external interface isn't being sent to our isp assigned
# ip address, drop it like a hot potato"
$IPT -A INPUT   -i $EXTIF -d ! $EXTIP  -j DROPl
```

## Tightening the internal LAN

We're going to apply the same logic to our internal LAN. In other words, any packets not originating from our predefined internal network will be rejected:

```
 # Now we will block internal addresses originating from anything but our
# two predefined interfaces.....just remember that if you jack your
# your laptop or another pc into one of these NIC's directly, you'll need
# to ensure that they either have the same ip or that you add a line explicitly
# that IP as well
# Interface one/internal net one
 $IPT -A INPUT   -i $INTIF1 -s ! $INTNET1 -j DROPI
 $IPT -A OUTPUT  -o $INTIF1 -d ! $INTNET1 -j DROPI
 $IPT -A FORWARD -i $INTIF1 -s ! $INTNET1 -j DROPI
 $IPT -A FORWARD -o $INTIF1 -d ! $INTNET1 -j DROPI
# Interface two/internal net two
 $IPT -A INPUT   -i $INTIF2 -s ! $INTNET2 -j DROPI
 $IPT -A OUTPUT  -o $INTIF2 -d ! $INTNET2 -j DROPI
 $IPT -A FORWARD -i $INTIF2 -s ! $INTNET2 -j DROPI
 $IPT -A FORWARD -o $INTIF2 -d ! $INTNET2 -j DROPI
```

Next we do some more Egress checking of outgoing packets and stop all icmp requests except for pinging:

```
 # An additional Egress check
 $IPT -A OUTPUT  -o $EXTIF -s ! $EXTNET -j DROPI
# Block outbound ICMP (except for PING)
 $IPT -A OUTPUT  -o $EXTIF -p icmp --icmp-type ! 8 -j DROPI
 $IPT -A FORWARD -o $EXTIF -p icmp --icmp-type ! 8 -j DROPI
```

Ok, where moving along now and we should test the script for errors.

## Ports

Assuming an all clear we're going to start plugging some of the more bothersome port holes:

```
 # COMmon ports:
```

```
# 0 is tcpmux; SGI had vulnerability, 1 is common attack
# 13 is daytime
# 98 is Linuxconf
# 111 is sunrpc (portmap)
# 137:139, 445 is Microsoft
# SNMP: 161,2
# Squid flotilla: 3128, 8000, 8008, 8080
# 1214 is Morpheus or KaZaA
# 2049 is NFS
# 3049 is very virulent Linux Trojan, mistakable for NFS
# Common attacks: 1999, 4329, 6346
# Common Trojans 12345 65535
 COMBLOCK="0:1 13 98 111 137:139 161:162 445 1214 1999 2049 3049 4329 6346 3128
8000 8008 8080 12345 65535"
# TCP ports:
# 98 is Linuxconf
# 512-515 is rexec, rlogin, rsh, printer(lpd)
#   [very serious vulnerabilities; attacks continue daily]
# 1080 is Socks proxy server
# 6000 is X (NOTE X over SSH is secure and runs on TCP 22)
# Block 6112 (Sun's/HP's CDE)
 TCPBLOCK="$COMBLOCK 98 512:515 1080 6000:6009 6112"

# UDP ports:
# 161:162 is SNMP
# 520=RIP, 9000 is Sangoma
# 517:518 are talk and ntalk (more annoying than anything)
 UDPBLOCK="$COMBLOCK 161:162 520 123 517:518 1427 9000 9 6346 3128 8000 8008
8080 12345 65535"
```

After defining the environment variables all we have to do is a  simple for loop to assign rules to them all:

```
  echo -n "FW: Blocking attacks to TCP port"
for i in $TCPBLOCK;
do
 echo -n "$i "
  $IPT -A INPUT   -p tcp --dport $i  -j DROPI
  $IPT -A OUTPUT  -p tcp --dport $i  -j DROPI
  $IPT -A FORWARD -p tcp --dport $i  -j DROPI
done
echo ""
  echo -n "FW: Blocking attacks to UDP port "
for i in $UDPBLOCK;
do
```

```
 echo -n "$i "
   $IPT -A INPUT   -p udp --dport $i  -j DROPI
   $IPT -A OUTPUT  -p udp --dport $i  -j DROPI
   $IPT -A FORWARD -p udp --dport $i  -j DROPI
done
echo ""
```

Ok, now with iptables each time we run the script it simply appends these to already existing chains...so things are probably getting a bit messy. For that reason we're going to jump to the beginning of our script....right after the enviroment variables for sed and grep, but before those of EXTIP and EXTBC and add a loop that deletes and flushes. This ensure we're working from a clean state. We didn't want to do that before because we couldn't have tested our script without either shutting down our connection or dropping our firewall completely. This script first sets all policies to DROP, than flushes and deletes our chains. In order to ensure that we can still ssh back into our server after a script restart we are going to append an INPUT chain for ssh. This should always be placed at the end of the script for now. This done in order to prevent a window from opening up while we reset rules which is a common error made:

```
  # Deny than accept: this keeps holes from opening up
# while we close ports and such
 $IPT        -P INPUT       DROP
 $IPT        -P OUTPUT      DROP
 $IPT        -P FORWARD     DROP
  # Flush all existing chains and erase personal chains
 CHAINS=`cat /proc/net/ip_tables_names 2>/dev/null`
 for i in $CHAINS;
 do
   $IPT -t $i -F
 done
 for i in $CHAINS;
 do
   $IPT -t $i -X
 done
 $IPT -A INPUT   -i $INTIF1 -p tcp --dport 22 --syn -m state --state NEW -j ACCEPT
```

## Sysctls

Right afterwards we are going to activate the sysctl's for tcp_syncookies, icmp_echo_ignore_broadcasts, rp_filter, and accept_source_route. Heretofore many of the rules we've been "testing" haven't been able to actually work. In essence we were simply doing syntax error tests. Now our rules will be "for real":

```
  echo 1 > /proc/sys/net/ipv4/tcp_syncookies
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
  # Source Address Verification
for f in /proc/sys/net/ipv4/conf/*/rp_filter;
do
 echo 1 > $f
done
# Disable IP source routing and ICMP redirects
for f in /proc/sys/net/ipv4/conf/*/accept_source_route;
do
 echo 0 > $f
done
for f in /proc/sys/net/ipv4/conf/*/accept_redirects;
do
 echo 0 > $f
done
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Now we're going to add ftp connection tracking so that we won't get PASV errors when emerging packages:

```
  # Opening up ftp connection tracking
MODULES="ip_nat_ftp ip_conntrack_ftp"
for i in $MODULES;
do
 echo "Inserting module $i"
 modprobe $i
done
```

## Basic Service NAT

Now back to end of our script, we are going to open up services for systems behind our firewall. I have included services such as IRC, MSN, ICQ, and NFS, FTP, domain, and time. And some others. The important thing to note is that these will ONLY be availabe BEHIND the firewall. So this will not enable someone to ftp into your LAN:

```
  IRC='ircd'
MSN=1863
ICQ=5190
NFS='sunrpc'
# We have to sync!!
```

```
PORTAGE='rsync'
OpenPGP_HTTP_Keyserver=11371
# All services ports are read from /etc/services
TCPSERV="domain ssh http https ftp ftp-data mail pop3 pop3s imap3 imaps imap2 time
$PORTAGE $IRC $MSN $ICQ $OpenPGP_HTTP_Keyserver"
UDPSERV="domain time"
echo -n "FW: Allowing inside systems to use service:"
for i in $TCPSERV;
do
 echo -n "$i "
 $IPT -A OUTPUT  -o $EXTIF -p tcp -s $EXTIP --dport $i --syn -m state --state NEW -j ACCEPT
 $IPT -A FORWARD -i $INTIF1 -p tcp -s $INTNET1 --dport $i --syn -m state --state NEW -j
ACCEPT
 $IPT -A FORWARD -i $INTIF2 -p tcp -s $INTNET2 --dport $i --syn -m state --state NEW -j
ACCEPT
done
echo ""
  echo -n "FW: Allowing inside systems to use service:"
for i in $UDPSERV;
do
 echo -n "$i "
 $IPT -A OUTPUT  -o $EXTIF -p udp -s $EXTIP --dport $i -m state --state NEW -j ACCEPT
   $IPT -A FORWARD -i $INTIF1 -p udp -s $INTNET1 --dport $i -m state --state NEW -j
ACCEPT
   $IPT -A FORWARD -i $INTIF2 -p udp -s $INTNET2
done
echo ""
```

Now we're done all that's left is allowing us to ping the  outside world by opening up pinging out of the firewall:

```
  # Allow to ping out
$IPT -A OUTPUT  -o $EXTIF -p icmp -s $EXTIP --icmp-type 8 -m state --state NEW -j ACCEPT
$IPT -A FORWARD -i $INTIF1 -p icmp -s $INTNET1 --icmp-type 8 -m state --state NEW -j
ACCEPT
$IPT -A FORWARD -i $INTIF2 -p icmp -s $INTNET2 --icmp-type 8 -m state --state NEW -j
ACCEPT
  # Allow firewall to ping internal systems
$IPT -A OUTPUT  -o $INTIF1 -p icmp -s $INTNET1 --icmp-type 8 -m state --state NEW -j
ACCEPT
$IPT -A OUTPUT  -o $INTIF2 -p icmp -s $INTNET2 --icmp-type 8 -m state --state NEW -j
ACCEPT
```

Now we are going to default to DROP and Log anything that's  left in case we overlooked

something. The ACCEPT entry's we made at the very beginning would come right before this in the final script:

```
 # Log & block whatever is left
$IPT -A INPUT            -j DROPl
$IPT -A OUTPUT           -j REJECTl
$IPT -A FORWARD          -j DROPl
```

And you're done. I had a friend nmap and nessus my connection with this rule set and as far as both of them were concerned the only thing it was even slightly sure about was that the ip existed...other than that nothing. I can IRC, MSN, ICQ, ane emerge sync to my hearts content.

PART III will cover setting up some essential SOHO services like NFS and CUPS in a security conscious manner.

## Scripts

A couple scripts.

### The full script

Now here's the full script from the above information in all its glory (I also put the ssh forwarding in a more appropriate place):

**File:** iptables.sh

```
     # First set LC_ALL to en to avoid l10n problems when awk-ing IPs etc.
export LC_ALL="en"
# External interface
EXTIF=ppp0
# Internal interface
INTIF1=eth1
INTIF2=eth2
```

```
# Loop device/localhost
LPDIF=lo
LPDIP=127.0.0.1
LPDMSK=255.0.0.0
LPDNET="$LPDIP/$LPDMSK"
# Text tools variables
IPT='/sbin/iptables'
IFC='/sbin/ifconfig'
G='/bin/grep'
SED='/bin/sed'
# Last but not least, the users
JAMES=192.168.1.77
TERESA=192.168.2.77
# Deny then accept: this keeps holes from opening up
# while we close ports and such
$IPT        -P INPUT       DROP
$IPT        -P OUTPUT      DROP
$IPT        -P FORWARD     DROP
# Flush all existing chains and erase personal chains
CHAINS=`cat /proc/net/ip_tables_names 2>/dev/null`
for i in $CHAINS;
do
 $IPT -t $i -F
done
for i in $CHAINS;
do
 $IPT -t $i -X
done
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
# Source Address Verification
for f in /proc/sys/net/ipv4/conf/*/rp_filter;
do
 echo 1 > $f
done
# Disable IP source routing and ICMP redirects
for f in /proc/sys/net/ipv4/conf/*/accept_source_route;
do
 echo 0 > $f
done
for f in /proc/sys/net/ipv4/conf/*/accept_redirects;
do
 echo 0 > $f
done
echo 1 > /proc/sys/net/ipv4/ip_forward
# Setting up external interface environment variables
```

```
EXTIP="`$IFC $EXTIF|$G addr:|$SED 's/.*addr:\([^ ]*\) .*/\1/'`"
#EXTBC="`$IFC $EXTIF|$G Bcast:|$SED 's/.*Bcast:\([^ ]*\) .*/\1/'`"
EXTBC="255.255.255.255"
EXTMSK="`$IFC $EXTIF|$G Mask:|$SED 's/.*Mask:\([^ ]*\)/\1/'`"
EXTNET="$EXTIP/$EXTMSK"
#echo "EXTIP=$EXTIP EXTBC=$EXTBC EXTMSK=$EXTMSK EXTNET=$EXTNET"
echo "EXTIP=$EXTIP EXTBC=$EXTBC EXTMSK=$EXTMSK EXTNET=$EXTNET"
# Due to absence of EXTBC I manually set it to 255.255.255.255
# this (hopefully) will serve the same purpose
# Setting up environment variables for internal interface one
INTIP1="`$IFC $INTIF1|$G addr:|$SED 's/.*addr:\([^ ]*\) .*/\1/'`"
INTBC1="`$IFC $INTIF1|$G Bcast:|$SED 's/.*Bcast:\([^ ]*\) .*/\1/'`"
INTMSK1="`$IFC $INTIF1|$G Mask:|$SED 's/.*Mask:\([^ ]*\)/\1/'`"
INTNET1="$INTIP1/$INTMSK1"
echo "INTIP1=$INTIP1 INTBC1=$INTBC1 INTMSK1=$INTMSK1 INTNET1=$INTNET1"
#Setting up environment variables for internal interface two
INTIP2="`$IFC $INTIF2|$G addr:|$SED 's/.*addr:\([^ ]*\) .*/\1/'`"
INTBC2="`$IFC $INTIF2|$G Bcast:|$SED 's/.*Bcast:\([^ ]*\) .*/\1/'`"
INTMSK2="`$IFC $INTIF2|$G Mask:|$SED 's/.*Mask:\([^ ]*\)/\1/'`"
INTNET2="$INTIP2/$INTMSK2"
echo "INTIP2=$INTIP2 INTBC2=$INTBC2 INTMSK2=$INTMSK2 INTNET2=$INTNET2"
# We are now going to create a few custom chains that will result in
# logging of dropped packets. This will enable us to avoid having to
# enter a log command prior to every drop we wish to log. The
# first will be first log drops the other will log rejects.
# Do not complain if chain already exists (so restart is clean)
$IPT -N DROPI   2> /dev/null
$IPT -A DROPI   -j LOG --log-prefix 'DROPI:'
$IPT -A DROPI   -j DROP
$IPT -N REJECTI 2> /dev/null
$IPT -A REJECTI -j LOG --log-prefix 'REJECTI:'
$IPT -A REJECTI -j REJECT
# Now we are going to accept all traffic from our loopback device
# if the IP matches any of our interfaces.
$IPT -A INPUT   -i $LPDIF -s   $LPDIP   -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $EXTIP   -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $INTIP1  -j ACCEPT
$IPT -A INPUT   -i $LPDIF -s   $INTIP2  -j ACCEPT
# Blocking Broadcasts
$IPT -A INPUT   -i $EXTIF  -d   $EXTBC   -j DROPI
$IPT -A INPUT   -i $INTIF1 -d   $INTBC1  -j DROPI
$IPT -A INPUT   -i $INTIF2 -d   $INTBC2  -j DROPI
$IPT -A OUTPUT  -o $EXTIF  -d   $EXTBC   -j DROPI
$IPT -A OUTPUT  -o $INTIF1 -d   $INTBC1  -j DROPI
$IPT -A OUTPUT  -o $INTIF2 -d   $INTBC2  -j DROPI
$IPT -A FORWARD -o $EXTIF  -d   $EXTBC   -j DROPI
```

```
$IPT -A FORWARD -o $INTIF1 -d   $INTBC1  -j DROPI
$IPT -A FORWARD -o $INTIF2 -d   $INTBC2  -j DROPI
# Block WAN access to internal network
# This also stops nefarious crackers from using our network as a
# launching point to attack other people
# iptables translation:
# "if input going into our external interface does not originate from our isp assigned
# ip address, drop it like a hot potato
$IPT -A INPUT   -i $EXTIF -d ! $EXTIP  -j DROPI
# Now we will block internal addresses originating from anything but our
# two predefined interfaces.....just remember that if you jack your
# your laptop or another pc into one of these NIC's directly, you'll need
# to ensure that they either have the same ip or that you add a line explicitly
# for that IP as well
# Interface one/internal net one
$IPT -A INPUT   -i $INTIF1 -s ! $INTNET1 -j DROPI
$IPT -A OUTPUT  -o $INTIF1 -d ! $INTNET1 -j DROPI
$IPT -A FORWARD -i $INTIF1 -s ! $INTNET1 -j DROPI
$IPT -A FORWARD -o $INTIF1 -d ! $INTNET1 -j DROPI
# Interface two/internal net two
$IPT -A INPUT   -i $INTIF2 -s ! $INTNET2 -j DROPI
$IPT -A OUTPUT  -o $INTIF2 -d ! $INTNET2 -j DROPI
$IPT -A FORWARD -i $INTIF2 -s ! $INTNET2 -j DROPI
$IPT -A FORWARD -o $INTIF2 -d ! $INTNET2 -j DROPI
# An additional Egress check
$IPT -A OUTPUT  -o $EXTIF -s ! $EXTNET -j DROPI
# Block outbound ICMP (except for PING)
$IPT -A OUTPUT  -o $EXTIF -p icmp --icmp-type ! 8 -j DROPI
$IPT -A FORWARD -o $EXTIF -p icmp --icmp-type ! 8 -j DROPI
# COMmon ports:
# 0 is tcpmux; SGI had vulnerability, 1 is common attack
# 13 is daytime
# 98 is Linuxconf
# 111 is sunrpc (portmap)
# 137:139, 445 is Microsoft
# SNMP: 161,2
# Squid flotilla: 3128, 8000, 8008, 8080
# 1214 is Morpheus or KaZaA
# 2049 is NFS
# 3049 is very virulent Linux Trojan, mistakable for NFS
# Common attacks: 1999, 4329, 6346
# Common Trojans 12345 65535
COMBLOCK="0:1 13 98 111 137:139 161:162 445 1214 1999 2049 3049 4329 6346 3128 8000 8008 8
# TCP ports:
# 98 is Linuxconf
# 512-515 is rexec, rlogin, rsh, printer(lpd)
```

```
#   [very serious vulnerabilities; attacks continue daily]
# 1080 is Socks proxy server
# 6000 is X (NOTE X over SSH is secure and runs on TCP 22)
# Block 6112 (Sun's/HP's CDE)
TCPBLOCK="$COMBLOCK 98 512:515 1080 6000:6009 6112"
# UDP ports:
# 161:162 is SNMP
# 520=RIP, 9000 is Sangoma
# 517:518 are talk and ntalk (more annoying than anything)
UDPBLOCK="$COMBLOCK 161:162 520 123 517:518 1427 9000"
echo -n "FW: Blocking attacks to TCP port "
for i in $TCPBLOCK;
do
  echo -n "$i "
  $IPT -A INPUT   -p tcp --dport $i  -j DROPI
  $IPT -A OUTPUT  -p tcp --dport $i  -j DROPI
  $IPT -A FORWARD -p tcp --dport $i  -j DROPI
done
echo ""
echo -n "FW: Blocking attacks to UDP port "
for i in $UDPBLOCK;
do
  echo -n "$i "
  $IPT -A INPUT   -p udp --dport $i  -j DROPI
  $IPT -A OUTPUT  -p udp --dport $i  -j DROPI
  $IPT -A FORWARD -p udp --dport $i  -j DROPI
done
echo ""
# Opening up ftp connection tracking
MODULES="ip_nat_ftp ip_conntrack_ftp"
for i in $MODULES;
do
 echo "Inserting module $i"
 modprobe $i
done
# Defining some common chat clients. Remove these from your accepted list for better security.
# ICQ and AOL are 5190
# MSN is 1863
# Y! is 5050
# Jabber is 5222
# Y! and Jabber ports not added by author and therefore left out of the script
IRC='ircd'
MSN=1863
ICQ=5190
NFS='sunrpc'
# We have to sync!!
```

```
PORTAGE='rsync'
OpenPGP_HTTP_Keyserver=11371
# All services ports are read from /etc/services
TCPSERV="domain ssh http https ftp ftp-data mail pop3 pop3s imap3 imaps imap2 \
        time $PORTAGE $IRC $MSN $ICQ  $OpenPGP_HTTP_Keyserver"
UDPSERV="domain time"
echo -n "FW: Allowing inside systems to use service:"
for i in $TCPSERV;
do
  echo -n "$i "
  $IPT -A OUTPUT  -o $EXTIF  -p tcp -s $EXTIP   --dport $i --syn -m state --state NEW -j ACCEPT
  $IPT -A FORWARD -i $INTIF1 -p tcp -s $INTNET1 --dport $i --syn -m state --state NEW -j ACCEPT
  $IPT -A FORWARD -i $INTIF2 -p tcp -s $INTNET2 --dport $i --syn -m state --state NEW -j ACCEPT
done
echo ""
echo -n "FW: Allowing inside systems to use service:"
for i in $UDPSERV;
do
  echo -n "$i "
  $IPT -A OUTPUT  -o $EXTIF  -p udp -s $EXTIP   --dport $i -m state --state NEW -j ACCEPT
  $IPT -A FORWARD -i $INTIF1 -p udp -s $INTNET1 --dport $i -m state --state NEW -j ACCEPT
  $IPT -A FORWARD -i $INTIF2 -p udp -s $INTNET2 --dport $i -m state --state NEW -j ACCEPT
done
echo ""
# Allow to ping out
$IPT -A OUTPUT  -o $EXTIF  -p icmp -s $EXTIP   --icmp-type 8 -m state --state NEW -j ACCEPT
$IPT -A FORWARD -i $INTIF1 -p icmp -s $INTNET1 --icmp-type 8 -m state --state NEW -j ACCEPT
$IPT -A FORWARD -i $INTIF2 -p icmp -s $INTNET2 --icmp-type 8 -m state --state NEW -j ACCEPT
# Allow firewall to ping internal systems
$IPT -A OUTPUT  -o $INTIF1 -p icmp -s $INTNET1 --icmp-type 8 -m state --state NEW -j ACCEPT
$IPT -A OUTPUT  -o $INTIF2 -p icmp -s $INTNET2 --icmp-type 8 -m state --state NEW -j ACCEPT
$IPT -A INPUT   -i $INTIF1 -p tcp --dport 22 --syn -m state --state NEW -j ACCEPT
$IPT -t nat -A PREROUTING  -j ACCEPT
$IPT -t nat -A POSTROUTING -o $EXTIF -s $INTNET1 -j MASQUERADE
$IPT -t nat -A POSTROUTING -o $EXTIF -s $INTNET2 -j MASQUERADE
$IPT -t nat -A POSTROUTING -j ACCEPT
$IPT -t nat -A OUTPUT -j ACCEPT
$IPT -A INPUT -p tcp --dport auth --syn -m state --state NEW -j ACCEPT
$IPT -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# Block and log what me may have forgot
$IPT -A INPUT   -j DROPl
$IPT -A OUTPUT  -j REJECTl
$IPT -A FORWARD -j DROPl
```

## Another iptables startup script

Another iptables startup script designed for a local linux router / linux server.

**File:** /etc/init.d/iptables

```
#! /sbin/runscript

depend() {
        before net
}

start() {
        ebegin "Starting firewall"

        modprobe ip_conntrack_ftp

        # reset all tables
        iptables -t filter -F
        iptables -t nat -F

        # delete all user chains
        iptables -t filter -X
        iptables -t nat -X

        # drop all packets by default
        iptables -t filter -P INPUT   DROP
        iptables -t filter -P FORWARD DROP
        iptables -t filter -P OUTPUT  DROP

        # create needed user-chains
        iptables -N drop_bad_packets

        # create a chain, that filters all bad packets:
        # drop all tcp-packets, that are not established and don't request a
        # new connection and drop every "unclean" package
        iptables -A drop_bad_packets -p tcp ! --syn -m state \
                --state NEW -j DROP
        iptables -A drop_bad_packets -m state --state INVALID -j DROP

        # do not accept, send or forward illegal packets
        iptables -A INPUT   -j drop_bad_packets
        iptables -A OUTPUT  -j drop_bad_packets
        iptables -A FORWARD -j drop_bad_packets

        # drop all packets that are comming from/going to 127.0.0.1 and
```

```
# are not going through loopback device
iptables -A INPUT  -i ! lo -s 127.0.0.1 -j DROP
iptables -A INPUT  -i ! lo -d 127.0.0.1 -j DROP
iptables -A OUTPUT -o ! lo -s 127.0.0.1 -j DROP
iptables -A OUTPUT -o ! lo -d 127.0.0.1 -j DROP

# allow masqueraded forwarding for eth0 device
for i in $MASQUERADINGDEVICES ; do
        iptables -t nat -A POSTROUTING -o "$i" -j MASQUERADE
done

# allow forwarding connections
for i in $FORWARDINGDEVICES ; do
        src="`echo $i | awk -F: '{ print $1 }'`"
        dest="`echo $i | awk -F: '{ print $2 }'`"

        if [ -z "$src" ] || [ -z "$dest" ] ; then
                ewarn "wrong FORWARDINGDEVICES entry (ignored): '$i'"
                continue
        fi

        iptables -A FORWARD -i "$src" -o "$dest" -j ACCEPT
        iptables -A FORWARD -i "$dest" -o "$src" -m state \
                 --state ESTABLISHED,RELATED -j ACCEPT
done

# allow all connections that are comming from or going to the
# loopback device and are from local host (other hosts wont be
# able to write to our loopback device)
iptables -A INPUT   -i lo -j ACCEPT
iptables -A OUTPUT  -o lo -j ACCEPT

for i in $NETDEVICES ; do
        # allow incoming data for established or related connections
        iptables -A INPUT -i "$i" -m state \
                --state ESTABLISHED,RELATED -j ACCEPT
        # allow incoming ping requests
        iptables -A INPUT -i "$i" -m state \
                --state NEW --protocol icmp \
                --icmp-type 8 -j ACCEPT
        # allow all outging data
        iptables -A OUTPUT -o "$i" -j ACCEPT
done



# allow all traffic from FULLACCESSIPS
```

```
        for i in $FULLACCESSIPS ; do
                device="`echo $i | awk -F: '{ print $1 }'`"
                ip="`echo $i | awk -F: '{ print $2 }'`"

                if [ -z "$ip" ] || [ -z "$device" ] ; then
                        ewarn "wrong FULLACCESSIPS entry (ignored): '$i'"
                        continue
                fi

                iptables -A INPUT -i "$device" -m state \
                                --state NEW \
                                --source "$ip" -j ACCEPT
        done

        # allow incomming connections to tcp ports
        for i in $INCOMINGPORTS ; do
                device="`echo $i | awk -F: '{ print $1 }'`"
                proto="`echo $i | awk -F: '{ print $2 }'`"
                ports="`echo $i | awk -F: '{ print $3 }'`"

                if [ -z "$device" ] || [ -z "$ports" ] || [ -z "$proto" ]; then
                        ewarn "wrong INCOMINGPORTS entry (ignored): '$i'"
                        continue
                fi
                iptables -A INPUT -i "$device" -m state -m multiport \
                        --state NEW --protocol "$proto" \
                        --dport "$ports" -j ACCEPT
        done

        # reject all other incoming data instead of just dropping them
        iptables -A INPUT -j REJECT

        echo 0 >/proc/sys/net/ipv4/tcp_ecn
        echo 1 >/proc/sys/net/ipv4/tcp_syncookies
        for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
                echo 1 >"$f"
        done

        # activate IP-Forwarding
        echo 1 >/proc/sys/net/ipv4/ip_dynaddr
        echo 1 >/proc/sys/net/ipv4/ip_forward

        eend 0
}

stop() {
```

```
        echo 0 >/proc/sys/net/ipv4/tcp_ecn
        echo 0 >/proc/sys/net/ipv4/tcp_syncookies
        echo 0 >/proc/sys/net/ipv4/ip_forward
        echo 0 >/proc/sys/net/ipv4/ip_dynaddr

        for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
                echo 0 >"$f"
        done

        ebegin "Stopping firewall"
                for a in `cat /proc/net/ip_tables_names`; do
                        /sbin/iptables -F -t "$a"
                        /sbin/iptables -X -t "$a"

                        if [ "$a" == nat ]; then
                                /sbin/iptables -t nat -P PREROUTING ACCEPT
                                /sbin/iptables -t nat -P POSTROUTING ACCEPT
                                /sbin/iptables -t nat -P OUTPUT ACCEPT
                        elif [ "$a" == mangle ]; then
                                /sbin/iptables -t mangle -P PREROUTING ACCEPT
                                /sbin/iptables -t mangle -P INPUT ACCEPT
                                /sbin/iptables -t mangle -P FORWARD ACCEPT
                                /sbin/iptables -t mangle -P OUTPUT ACCEPT
                                /sbin/iptables -t mangle -P POSTROUTING ACCEPT
                        elif [ "$a" == filter ]; then
                                /sbin/iptables -t filter -P INPUT ACCEPT
                                /sbin/iptables -t filter -P FORWARD ACCEPT
                                /sbin/iptables -t filter -P OUTPUT ACCEPT
                        fi
                done
        eend $?
}
```

**Warning:** The following file is an **EXAMPLE - DO NOT USE IT WITHOUT MODIFICATION** **File:** /etc/conf.d/iptables

```
        # open incoming ports (device:proto:ports)
INCOMINGPORTS="eth0:tcp:22,80,443,1500-1510 eth0:udp:53"

# allow data connections through these devices
NETDEVICES="eth0 ppp0 tun1"

# allow forwardings on these devices (src:dest)
FORWARDINGDEVICES="eth0:ppp0 eth0:tun1"

# masquerade ips on these devices
```

```
MASQUERADINGDEVICES="ppp0"

# allow all traffic from these ips (device:ip)
FULLACCESSIPS="eth0:192.168.0.4"
```

## Another script

**File:** firewall.sh

```
#!/bin/bash
IPTABLES='/sbin/iptables'
# Set interface values
EXTIF='ppp0'
INTIF1='eth1'
INTIF2='eth2'
# enable ip forwarding in the kernel
/bin/echo 1 > /proc/sys/net/ipv4/ip_forward
# flush rules and delete chains
$IPTABLES -F
$IPTABLES -X
# enable masquerading to allow LAN internet access
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j MASQUERADE
# forward LAN traffic from $INTIF1 to Internet interface $EXTIF
$IPTABLES -A FORWARD -i $INTIF1 -o $EXTIF -m state --state NEW,ESTABLISHED -j ACCEPT
# forward LAN traffic from $INTIF2 to Internet interface $EXTIF
$IPTABLES -A FORWARD -i $INTIF2 -o $EXTIF -m state --state NEW,ESTABLISHED -j ACCEPT
#echo -e "       - Allowing access to the SSH server"
$IPTABLES -A INPUT --protocol tcp --dport 22 -j ACCEPT
#echo -e "       - Allowing access to the HTTP server"
$IPTABLES -A INPUT --protocol tcp --dport 80 -j ACCEPT
# block out all other Internet access on $EXTIF
$IPTABLES -A INPUT -i $EXTIF -m state --state NEW,INVALID -j DROP
$IPTABLES -A FORWARD -i $EXTIF -m state --state NEW,INVALID -j DROP
```

**Note:** This script was written by someone on another forum....I've since lost the address to the thread or forum..., but thanks goes to him.

This script can be found on HOWTO Iptables and stateful firewalls as well.

## Reset iptables

This will make iptables accept everything:

```
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X
```

# Hardware Related Information

## Interface configuration

The author created this document is using a PPPoE connection to the internet and the 2.6.x kernel.

However, the only adaptation that would need to be made is to replace 'ppp0' with 'eth0' (or whatever NIC you have facing the internet -- this will become clear later).

In my set up, I have three NIC's, one is connected to the WAN through PPPoE, the other two to my internal network. In order for them all to play nicely with iptables and masquerading (NAT'ing), they must be set to different subnets. For example, the two NIC's connected to my internal computers, e.g., the "internal NIC's", are assigned: 192.168.1.1 and 192.168.2.1 respectively. It should be noted here that it is perfectly acceptable to connect these internal NIC's to any network capable device, such as a switch or hub. For pppoe connections we make sure the NIC connected to the outside world, e.g. the external NIC is not assigned any ip....its entries in /etc/conf.d/net should be left blank. This is due to pppoe acting as a virtual device which handles bringing up the NIC. We must also assign proper netmasks and broadcast values to these interfaces. Your /etc/conf.d/net should look like these:

Server:

**File:** /etc/conf.d/net

```
# For pppoe connections you do not want to set values for eth0, simply add
# net.ppp0 or rp-pppoe to your default runlevel
config_eth1=( "192.168.1.1 broadcast 192.168.1.255 netmask 255.255.255.0" )
config_eth2=( "192.168.2.1 broadcast 192.168.2.255 netmask 255.255.255.0" )
```

Client One:

**File:** /etc/conf.d/net

```
config_eth0=( "192.168.1.77 broadcast 192.168.1.255 netmask 255.255.255.0" )
gateway_eth0=( "default via 192.168.1.1" )
```

Client Two:

**File:** /etc/conf.d/net

```
config_eth0=( "192.168.2.77 broadcast 192.168.2.255 netmask 255.255.255.0" )
gateway_eth0=( "default via eth0/192.168.2.1" )
```

The gateways for the clients are set to the internal ip's of the NIC on the server as should be expected. Now add all the interfaces to the default run level and restart connections:

## Start network cards

```
rc-update add net.eth1 default; rc-update add net.eth2 default; rc-update add net.ppp0 default;
/etc/init.d/net.eth1 restart; /etc/init.d/net.eth2 restart; /etc/init.d/net.ppp0 restart;
```

Now verify that you are connected to the internet on the server machine (the clients will not be.....yet) and that all the interfaces can ping each other.

## Server

```
ping www.google.com;
```

ping 192.168.1.78
ping 192.168.2.78
ping 192.168.1.77
ping 192.168.2.77

Next ensure that your clients have appropriate DNS's set in your /etc/resolv.conf.

# Troubleshooting

## Failure on COMMIT lines

iptables failures will nearly always occur on the COMMIT lines. This is because while the rules are read earlier, iptables does not check the match (-m) option until this point. The actual error will be somewhere between the COMMIT line where the error occurred and the previous COMMIT line.

The first things to check for when you get errors are typos and that you have the correct modules loaded (if you're unsure, the easiest way is to build all the features into the kernel - you might want to skip features marked experimental for this as they may cause issues and you're unlikely to be using them).

One way to locate the line causing the problem is to disable all your rules (this can be done by commenting them out by placing a # at the beginning of the line) and then re-enabling them one-by-one until the error occurs.

Another way of finding the erroneous line is to run your firewall script with the bash parameter -x. This will cause the bash shell to print every line of the script before executing it and makes debugging quite easy:

```
bash -x firewall.sh
```

This will output something like this:

```
  + IPT=/sbin/iptables
+ /sbin/iptables -F
+ /sbin/iptables -P INPUT DOP
iptables: Bad policy name
+ /sbin/iptables -P OUTPUT DROP
[...]
```