Global Information Assurance Certification Paper

Table of Contents

Summary/Abstract

Topic: This paper outlines my steps in converting a small business from a router-based perimeter to a configuration offering more 'defense in depth'. My goal for this paper is to provide a top-level view of the steps required to build, install, and maintain a robust and secure Linux firewall. I formatted the paper as a resource for step-by-step instructions on the various components of a firewall installation. I've also included administration tasks and ongoing responsibilities for firewall maintenance. Detailed software installation instructions are not typed into this document, since they are readily available from other sources. I've provided links to each of these sources where appropriate.

Before the addition of a Firewall

Overview: Our small business network contains approximately 40 PCs connected to the Internet through an ascend router. The Ascend router provided filtering on IP address, but was limited to 12 filters. Our business needs require remote access for clients and employees using Secure Shell (SSH). The business outgrew the 12-filter limit. The Ascend router filters work like this:

Source IP	Source Port	Redirect to
x.x.x.x	22	192.168.1.x
x.x.x.x	22	192.168.1.x
x.x.x.x	2220	192.168.1.x
etc		

The router did a good job of restricting access to certain IP/ports, but did not provide enough filters to support the number of clients and employees who need access. There were no outbound filters in place, so the network was vulnerable to leaking private IP packets out to the Internet, and/or participating inadvertently in outbound attacks. The router provided no protection from spoofing, malformed packets, invalid port requests, etc. In addition, the Ascend router provided no logging capabilities, so we were unable to determine what traffic the router was stopping.

Since the router was the only "door" to the Internet, we had no additional defenses to protect our network from the outside. Additionally, the 12-filter limit forced us to make our filtering rules more open (so we could allow more people through without exceeding 12 rules). For example, we originally filtered on incoming IP/source port and redirected the allowable traffic to the appropriate server (as in the above sample). Over time, we had to drop the incoming IP and filter only on source port to pass the traffic through to the proper server.

At the time we decided something had to change, our Ascend router configuration looked like this:

Source IP	Source Port	Redirect to
Any	22	192.168.1.x
Any	2220	192.168.1.x
etc		

Obviously, this filtering strategy is not as secure as it was in the past. We moved all source IP filtering to the individual servers rather than letting the gateway handle it, which adds unnecessary network traffic (in addition to the security risks). We are running tcpwrappers on the servers to control access by source IP, but removing the source IP filter from the router puts possible bad guys directly on our server before we check them.

Solution: Replace the Ascend router with a Cisco router that provides more filtering capabilities and add a firewall to provide an additional layer and more flexibility. Several firewall packages were considered, including: Raptor and CheckPoint. However, due to budget constraints (the business provided only about $3000 for this project), the following equipment was selected:

- Cisco 1000 Series Router – purchased, installed and configured by our ISP
- Linux Firewall running ipchains on existing Pentium III processor with dual hard drives (one disk drive used for backups)

Note: this project began as a packet-filtering firewall project. After establishing the packet-filtering firewall (ipchains), I converted the firewall to iptables for stateful filtering. This paper covers installation of ipchains, as well as conversion to iptables.

Step-by-Step project outline

Cisco Router Configuration

Our ISP installed and configured our Cisco Router based on the following rules (which were provided in my SANS Training):

1. Block all private IP addresses in and out
2. Block IP spoofing
3. Only allow ICMP request or reply
4. Block all broadcast IPs in and out
5. Block incoming IP with destination IP not equal to router or firewall address (internal network uses private, non-routable IP addresses)
6. Block all non IP traffic
7. TCP/UDP packets with a source port of zero

I'm able to verify the rules are working properly by checking my firewall logs (more on that later). None of the packets listed in the rules above should reach my firewall.

Linux Firewall setup

Hardware and Operating System Installation

First, the PC required an additional network card and a Linux installation. I chose RedHat 7.x as the OS, inserted the network card and installed RedHat from CD. All hardware was recognized without a problem.

One of the first decisions you'll need to make during installation is to select the type of machine (workstation, server, custom, etc). Selecting 'workstation' gives you limited services. On the other hand, selecting 'server' gives you every service available (web server, DNS, news, mail, etc). Since this machine will be a firewall, we need to be careful not to install unnecessary services. During the install, select 'custom' as the system type. This gives you the most flexibility in choosing the software to install. You can click the box marked 'choose software packages' to see exactly what packages will be installed. Make sure OpenSSH is included in your installation if you plan to administer the machine remotely. OpenSSH provides an encrypted, telnet-like connection. You'll also want to select tripwire (discussed in more detail later).

Another decision you'll be faced with is whether to run your machine in graphical mode. I choose *not* to boot into graphical mode to cut down on the number of processes running. A firewall, if running correctly, shouldn't need much attention other than monitoring and patching. If you're comfortable at the Unix command line (and you really should be if you're planning to administer a Linux firewall yourself), then you don't need the graphical interface. My recommendation is to boot to normal network mode (not graphical mode) and start the graphical interface when/if you need it by running startx at the Unix command line.

When prompted, create a boot disk copy, label it, and keep it someplace safe.

The RedHat installation prompts you through everything you need to configure, so I won't go into great detail here. If you need help, check out the RedHat installation pages at:
http://www.redhat.com/support/resources/install_upgrade/installing_linux.html

Software Setup

Kernel Changes

RedHat 7.x will give you an option to select 'firewalling' during the install. If you say 'yes', the kernel will be built with ipchains enabled. To use an existing installation (you already have RedHat loaded onto a machine), you'll need to make some changes to the Linux kernel. If you've never changed a Linux kernel before, check out the kernel HOWTO at: www.tldp.org/HOWTO/Kernel-HOWTO.html. To enable ipchains on your kernel, click on 'Networking Options' and enable 'network firewall', 'IP: multicasting', "IP: firewalls', and 'IP: masquerading'. When finished, select the option to save the kernel.

To rebuild the kernel, run 'make dep clean bzImage modules'. On RedHat 7.x, it automatically saves the old kernel before rebuilding the new. After rebooting, you will see both kernels as options. In the event of problems with the new kernel, simply select the old kernel at reboot, fix the problem and rebuild.

RedHat 7.x with up2date

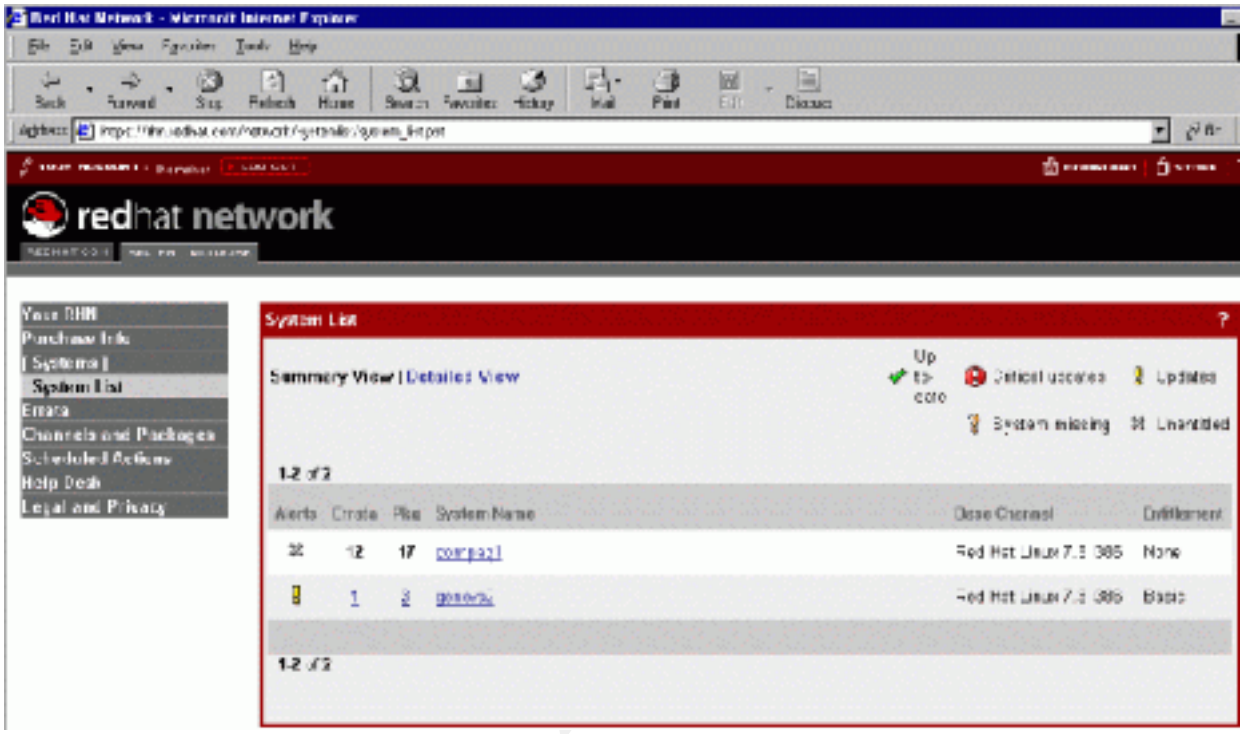
I registered the system with RedHat's RHN (RedHat Network) for use of the up2date utility. Up2date is a tool that simplifies system administration of patches and updates (and I highly recommend it). All transactions to RHN are encrypted, the site uses digital certificates, and all electronic communications from RHN are signed using GPG. For more information on RHN security and quality assurance, see: http://www.redhat.com/docs/manuals/RHNetwork/ref-guide/security-note.html.

Up2date can be used as a command-line tool and run when you want, or it can be run as a daemon – constantly checking RHN for updates and loading them automatically. For the firewall, I chose to have RHN automatically *notify* me, via email, of any new alerts and updates. Then, I *manually* run the up2date tool for the upgrades. This combination gives me real-time notification AND control of the installation process. Since some updates make changes to configuration files, I prefer to run them manually so I can check the update immediately. On my Linux workstations, however, I've run the up2date daemon with great success.

Here are some details on RHN up2date:

As a RedHat user, you get one free entitlement. If you register your new system during the installation, your entitlement will be activated. If you skipped that part of the installation, visit http://rhn.redhat.com/network to entitle your RedHat system for updates. If you manage multiple RedHat machines, you can purchase additional entitlements for $60 per system. By purchasing additional entitlements under your registration name, you can view/manage your entire network of

servers from one web page. Here's an example of the RedHat Network 'System List' page:



When you register, your system sends its software list to the RedHat network. When security and bugfix announcements are made, your system status will change from "up to date" status to either "updates" or "critical updates". As a side note: on systems running a graphical interface (gnome, KDE, etc), the bottom right corner of the screen shows an icon giving the status of updates for this machine. The following graphical icons apply:

 critical updates icon

 'up to date' icon

 'updates' icon

Keeping your RedHat system up-to-date

If you are not running the up2date daemon (which automatically loads the updates for you), the up2date process can be started through the graphical interface (run 'update agent' from the Gnome icon), or simply type 'up2date –u' at the command line.

After installing new software or removing existing software, you must send a new profile to the RHN so your notifications remain accurate. To send a new profile to RHN, type 'up2date –p' at the command line.

System Hardening

An important part of any installation is removing unnecessary services and open ports. The services running on your firewall will vary depending on the selections you made during the installation. Linux runs services from several locations.

inetd services:

These are typically services that are called as needed. They don't run all the time, but quickly load up when a request is received. Examples of inetd services include telnet, ftp, finger, and time. On earlier versions of Linux, these were found in one file: /etc/inetd.conf. Newer versions have a separate script for each service. These scripts are stored in the /etc/xinetd.d directory.

Check through /etc/xinetd.d scripts and disable all unnecessary services, such as: finger, ntalk, rlogin, rsh, talk, and telnet. To disable a service in the xinetd.d directory, add the line 'disable = yes' to the end of the script. On my firewall, I've disabled all xinetd.d scripts.

Start-up scripts:

Next, check all start-up scripts. Start-up scripts can be found in /etc/rc.d/rc3.d if you are running in regular network-mode, or /etc/rc.d/rc5.d if you are running in graphical network mode. All start-up scripts begin with "S".

To display the status of a process, use the chkconfig command `--list` option. For example, you can check the status of the RedHat up2date daemon as follows:

```
chkconfig --list rhnsd
rhnds          0:off   1:off   2:off   3:on    4:on    5:on  6:off
```

To turn off a process, use the chkconfig command:

```
chkconfig --level 345 rhnsd off
```

After you've turned unnecessary scripts off, reboot (to make sure everything stays off) and run 'netstat –a'. This will show all open ports, for example:

```
[lhomsher@hostname rc3.d]$ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
State
tcp   0     0 *:sunrpc                *:*         LISTEN
tcp   0     0 *:2224                  *:*         LISTEN
tcp   0     0 loopback:smtp           *:*         LISTEN
```

```
tcp   0     0 hostname:2224 192.168.x:1068  ESTABLISHED
tcp   0     0 *:32768             *:*         LISTEN
udp   0     0 *:sunrpc            *:*
```

The netstat above shows sunrpc running, which is a networking technology developed by Sun Microsystems. According to Internet Security Systems, RPC can be very dangerous when exposed to the Internet. For more information, see: (http://www.iss.net/security_center/advice/Services/SunRPC/default.htm). On our firewall, we can stop this service using the chkconfig command. The other ports are 2224 (our ssh port) and smtp (sendmail), used for email. We'll need both of these open on this machine.

We also see port 32768 open, which is rpc.statd, used for NFS server notification. Since I'm not using this machine as an NFS server, I don't need this process. In order to shut down this port, I had to turn off nfslock, which I did using the chkconfig command. How did I know that nfslock turns off rpc.statd? Run the grep command against the rc3.d (the start-up directory) to find any script that contains rpc.statd. You can see by the example below that rpc.statd is tied to the nfslock service:

```
[lhomsher@hostname rc3.d]$ grep rpc.statd *
S86nfslock:[ -x /sbin/rpc.statd ] || exit 0
S86nfslock:     daemon rpc.statd
K86nfslock:     killproc rpc.statd
K86nfslock:     status rpc.statd
K86nfslock:     /sbin/pidof rpc.statd >/dev/null 2>&1; STATD="$?"
```

To turn it off, run the following command:

```
chkconfig --level 345 nfslock off
```

After cleaning up all unnecessary services and processes, our open ports show only those services we need for this box:

```
[root@hostname rc3.d]# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
State
tcp        0      0     *:2224              *:*                 LISTEN
tcp        0      0     loopback:smtp       *:*                 LISTEN
tcp        0    268   hostname:2224 192.168.1.x:1068 ESTABLISHED
```

Set up logging

Linux firewalls provide great logging features. In your ipchains or iptables rules, you specify what packets to log. Your firewall logs can show you all rejected traffic (deny, reject, and drop are discussed later), as well as selected accepted traffic. Traffic that is rejected by your firewall can often give you an indication of an attempted attack or 'rattling of the doorknobs' (probing for information).

In addition to the firewall logs, RedHat provides plenty of good system logging, such as boot messages, mail messages, information on daemons, authentication information, etc. The level of detail you will see in your log files is a result of the settings in syslog.conf (the configuration file for system logging). I find the RedHat 7.x /etc/syslog.conf file default settings work well for my use. However, you can change the default settings if necessary. There is documentation at the top of the file to explain the settings. If you are fortunate enough to have a centralized logging machine, you'll want to change /etc/syslog.conf to take advantage of it. By logging to another server, you make it a little harder for an attacker to cover his tracks.

The default logging also includes a logwatch cron (scheduled) job, which sends all logon information and other interesting tidbits to root's mailbox.

Set up email notification
The default installation sends all notification to 'root'. If you'd like to receive this information via email, you'll need to change the: /etc/aliases file to redirect root to your email address as follows:

```
# Person who should get root's mail
root:   lhomsher@yourdomain.com
```

Set up backups
I set up a simple, automated backup script using tar, gzip and the cron daemon. First, I decided which directories I want to backup daily/weekly/monthly, creating a separate script file for each schedule. The script can contain something like:

```
tar -c -G -v -f /backup/backup_etc.tar /etc
gzip -9 -f /backup/backup_etc.tar
```

The lines above create a compressed tar archive. After creating a script for daily/weekly/monthly backups, create a cron job in /etc/cron.daily, /etc/cron.weekly, and/or /etc/cron.monthly. Here is an example:

```
/pathtoscript/backup.files 1>/pathtologs/backup.files.txt \
2>/pathtologs/backup.files.errors
```

The redirect of output sends the backup logs to log files instead of the screen (or email if your console output is redirected in syslog.conf).

On my firewall, I backup all directories weekly to a 2nd hard disk.

A little background on cron, the unix scheduler
(for more information on cron, see: http://www.uwsg.iu.edu/usail/automation/cron.html or type 'man cron' to see the cron 'man' (documentation pages).

cron uses /etc/crontab to determine what months, days, weeks, hours, and minutes to execute the directories listed in the crontab file. Here is a typical /etc/crontab:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

The rather cryptic numbers and stars at the beginning of the run-parts lines have the following meanings:

```
* Minutes (0-59)
* Hours (0-23)
* Days (1-31)
* Months (1-12)
* Days of the week (0-6, or Sun, Mon, Tue, etc.)
```

So, the line corresponding to:

- /etc/cron.hourly says run the first minute of every hour of every day of every month.
- /etc/cron.daily say run the 2nd minute in the 4th hour of every day of every month.
- /etc/cron.weekly says run the 22nd minute in the 4th hour on the Sunday of every week.
- You get the idea.

You shouldn't need to change the crontab file unless you need to run a process on a schedule other than what is provided by RedHat. To change the crontab, be sure to use the 'crontab –e' command – not your editor! It's also a good idea to modify the /etc/cron.allow and /etc/cron.deny files to control who is allowed to change the crontab file.

To run your backup each week, place your backup script in the /etc/cron.weekly directory. Cron is also used by tripwire (discussed next) to send daily integrity reports.

Protecting against unauthorized changes to the system

Tripwire is "Data Integrity Assurance" software (provided on the RedHat 7.x CD or available from http://www.tripwire.org/). It will take a 'snapshot' of the system and report on any changes. This is important software for your firewall and any other critical servers. Without data integrity software, you may never know if a rootkit or backdoor has been installed on your system. A rootkit is typically software installed on your system without your knowledge. Sometimes, its

purpose is to obtain root access or operate a network sniffer. Often, a rootkit will replace system binaries with its own copies, which is where tripwire comes in. Tripwire will alert you of any changes to system binaries, directories, or other critical files you define during the configuration.

To install tripwire, run the tripwire script `/etc/tripwire/twinstall.sh`

For complete instructions, see the tripwire documentation included in the installation. You can also find documentation at:
http://sourceforge.net/projects/tripwire and
http://www.tripwire.org/resources/index.php

Here is a summary containing a few additional hints, some not found in the tripwire manual:

After installation, you'll need to modify the policy file to meet your system's needs. The policy file is normally /etc/tripwire/twpol.txt. This process can be tedious, but it's very important that the final policy file accurately reflects your system files.

If you want email notification of integrity problems, add the 'emailto' field to the 'rulename' lines. For example:

```
(
  rulename = "Tripwire Binaries",
  severity = $(SIG_HI), emailto=lhomsher@yourdomain.com
)
```

If this is a first-time installation of tripwire, you'll need to build the database file by running:

```
tripwire -m p twpol.txt
```

When your policy text file (twpol.txt) is ready to go, run the following command to create the actual policy (tw.pol).

```
twadmin -m P /etc/tripwire/twpol.txt
```

For security reasons, the tripwire policy file (tw.pol) is encrypted. After you are finished with your configuration, remove the plain-text version (twpol.txt) of the policy file by backing it off the system to diskette or tape.

During installation, a tripwire-check script is added to the /etc/cron.daily directory. The script performs an integrity check every day. If you've configured tripwire with proper email functionality, the integrity report will be delivered to your mailbox.

From time-to-time, you'll need to update your policy file (for example, after installing updates or adding/removing software). Here are the steps required to update your policy file to reflect the current system environment:

1. Restore and change your plain-text policy file as needed.
2. Run the tripwire update command:
   `tripwire -m p /etc/tripwire/twpol.txt`

Or you can recreate your policy from a particular integrity report (if you are certain the integrity report is correct) by running the following command:
`tripwire -m u -r /var/lib/tripwire/report/hostname-xxxxxx-xxxxx.twr`

Again, when you are finished with your changes, back off the plain-text version of the policy file: twpol.txt

Installing ipchains

Now, we're finally ready to begin configuring our ipchains. Rusty Russell has a great HOWTO guide on ipchains at: http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html. In addition to this HOWTO, I purchased a copy of the book *Linux Firewalls* by Robert Ziegler, which gives much more detail on how to set up a firewall. It also includes a nice chapter on how to read the firewall logs (yes, if you install a firewall, *someone* will need to review the log files).

I downloaded my starting script from http://rcf.mvlan.net:8080/ . My original script was rc.firewall, which is now available as rcf.

Our network requires the following considerations:

- Clients require remote access into the application server. This is accomplished using OpenSSH on port 22.
- IT Staff requires remote access into all servers. This is accomplished by running OpenSSH on a different port for each server. The firewall rules forward the incoming request to the appropriate server based on port number. All of our servers requiring remote access are running some flavor of Unix, but I believe there is an OpenSSH available for the Windows platform as well.
- There is an ftp server provided to authorized users only. The firewall must forward all port 20 and port 21 requests to the ftp server. The ftp server requires a valid logon and also requires an entry in the /etc/ftphosts file.
- The ISP manages employee email accounts. Incoming POP3 email is passed through the firewall and delivered to the user's desktop. Norton Corporate Edition AV is running on Windows 2000 server and pushes updates to the desktops.
- The ISP handles DNS queries for the business. There is no internal DNS server.
- The internal web server is not accessible from outside the network.
- Outbound activity is filtered to prevent netbios and other internal-network activity from escaping to the Internet.

Our firewall policy is to "deny all except what is explicitly allowed". This is done with the following rules:

```
# Flush any existing rules from all chains
ipchains -F

# Set the default policy to deny/reject
ipchains -P input   DENY -l
ipchains -P output  REJECT -l
ipchains -P forward REJECT -l
```

The rest of the firewall script sets up variables and runs various ipchains commands to specifically allow input, output, and forward traffic. Here are the details of how we configured the firewall to meet our network's needs:

```
#  We need to add a module for ftp:
/sbin/modprobe ip_masq_ftp

# Set masquerade timeout to 10 hours for TCP connections.
ipchains -M -S 36000 0 0

# Unlimited traffic on the loopback interface
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT

# Allow certain types of ICMP:
# (4)  Source_Quench (type 4)
#      incoming & outgoing requests to slow down (flow control)
ipchains -A input  -i $EXTERNAL_INTERFACE -p icmp \
          -s $ANYWHERE 4 -d $IPADDR -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 4 -d $ANYWHERE -j ACCEPT

# (12) Parameter_Problem (type 1 incoming & outgoing error msgs
 ipchains -A input  -i $EXTERNAL_INTERFACE -p icmp \
         -s $ANYWHERE 12 -d $IPADDR -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 12 -d $ANYWHERE -j ACCEPT

# (3)  Dest_Unreachable, Service_Unavailable (type 3)
#      incoming & outgoing size negotiation, service or
#      destination unavailability, final traceroute response
ipchains -A input  -i $EXTERNAL_INTERFACE -p icmp \
         -s $ANYWHERE 3 -d $IPADDR -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 3 -d $MY_ISP -j ACCEPT
# added for ISP DNS servers (was getting log entries)
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 3 -d $NAMESERVER_1 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 3 -d $NAMESERVER_2 -j ACCEPT
```

```
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 3 -d $NAMESERVER_3 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR fragmentation-needed -d $ANYWHERE -j ACCEPT

# allow outgoing pings to anywhere
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp \
         -s $IPADDR 8 -d $ANYWHERE -j ACCEPT

ipchains -A input  -i $EXTERNAL_INTERFACE -p icmp \
         -s $ANYWHERE 0 -d $IPADDR -j ACCEPT

# DNS client modes (53) – using ISP DNS servers
# ---------------------
ipchains -A output -i $EXTERNAL_INTERFACE -p udp \
         -s $IPADDR $UNPRIVPORTS \
         -d $NAMESERVER_1 53 -j ACCEPT
ipchains -A input  -i $EXTERNAL_INTERFACE -p udp \
         -s $NAMESERVER_1 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p udp \
         -s $IPADDR $UNPRIVPORTS \
         -d $NAMESERVER_2 53 -j ACCEPT
ipchains -A input  -i $EXTERNAL_INTERFACE -p udp \
         -s $NAMESERVER_2 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p udp \
         -s $IPADDR $UNPRIVPORTS \
         -d $NAMESERVER_3 53 -j ACCEPT
ipchains -A input  -i $EXTERNAL_INTERFACE -p udp \
         -s $NAMESERVER_3 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# Sending Mail through a remote SMTP gateway (25)
# SMTP client to an ISP account without a local server
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS \
         -d $SMTP_GATEWAY 25 -j ACCEPT
ipchains -A input  -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $SMTP_GATEWAY 25 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# POP (110) - Retrieving Mail as a POP Client
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS \
         -d $POP_SERVER 110 -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $POP_SERVER 110 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

```
# SSH (22) - Allowing Access to local SSH servers
#lh: allow only specific clients in:
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp \
         -s $MY_CLIENT3 $UNPRIVPORTS \
         -d $IPADDR 22 -j ACCEPT -l
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $IPADDR 22 \
         -d $MY_CLIENT3 $UNPRIVPORTS -j ACCEPT
(repeat for all SSH ports)

# FTP (20, 21) - Allowing Incoming Access to Your Local FTP Svr
# incoming request
ipchains -A input  -i $EXTERNAL_INTERFACE -p tcp \
         -s $ANYWHERE $UNPRIVPORTS \
         -d $IPADDR 21 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $IPADDR 21 \
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
# Normal Port Mode FTP Data Channel Responses
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR 20 \
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
ipchains -A input  -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $ANYWHERE $UNPRIVPORTS \
         -d $IPADDR 20 -j ACCEPT

# Here's where we allow outbound requests:
# HTTP (80) - Accessing Remote Web Sites as a Client
# ------------------------------------------------

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 80 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $ANYWHERE 80 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# FTP (20, 21) - Allowing Outgoing Client Access to Remote FTP Servers
# ------------------------------------------------------------------

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 21 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
         -s $ANYWHERE 21 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# HTTPS (443) - Accessing Remote Web Sites Over SSL as a Client
# -----------------------------------------------------------

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS \
```

```
        -d $ANYWHERE 443 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
        -s $ANYWHERE 443 \
        -d $IPADDR $UNPRIVPORTS -j ACCEPT

# HTTP Proxy client (8008/8080)
# -----------------------------

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
        -s $IPADDR $UNPRIVPORTS \
        -d $WEB_PROXY_SERVER $WEB_PROXY_PORT -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
        -s $WEB_PROXY_SERVER $WEB_PROXY_PORT \
        -d $IPADDR $UNPRIVPORTS -j ACCEPT

# WHOIS client (43)
# -----------------
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp \
        -s $IPADDR $UNPRIVPORTS \
        -d $ANYWHERE 43 -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y \
        -s $ANYWHERE 43 \
        -d $IPADDR $UNPRIVPORTS -j ACCEPT

# TRACEROUTE
# traceroute usually uses -S 32769:65535 -D 33434:33523
# ----------------------------------------------------

# Enabling Outgoing traceroute Requests
# -------------------------------------

ipchains -A output -i $EXTERNAL_INTERFACE -p udp \
        -s $IPADDR $TRACEROUTE_SRC_PORTS \
        -d $ANYWHERE $TRACEROUTE_DEST_PORTS -j ACCEPT

# incoming query from the ISP.
# All others are denied by default.
# --------------------------------

ipchains -A input -i $EXTERNAL_INTERFACE -p udp \
        -s $MY_ISP 32769:65535 \
        -d $IPADDR 33434:33523 -j ACCEPT

# Unlimited traffic within the local network.
# All internal machines have access to the fireall machine.

# Masquerade internal traffic.
# Turn on IP forwarding
echo "1" > /proc/sys/net/ipv4/ip_forward

# All internal traffic is masqueraded externally.
ipchains -A forward -i $EXTERNAL_INTERFACE -s $LAN_1 -j MASQ
ipchains -A input  -i $LAN_INTERFACE_1 \
        -s $LAN_1 -j ACCEPT
```

```
ipchains -A output -i $LAN_INTERFACE_1 \
         -d $LAN_1 -j ACCEPT

# Added ipmasqadm lines to provide port forward to the
appropriate servers:
            ipmasqadm portfw -a -P tcp -L $IPADDR 22 -R 192.168.1.x 22
            ipmasqadm portfw -a -P tcp -L $IPADDR 2221 -R 192.168.1.x 2221
            ipmasqadm portfw -a -P tcp -L $IPADDR 2220 -R 192.168.1.x 2220
            ipmasqadm portfw -a -P tcp -L $IPADDR 2223 -R 192.168.1.x 2223
            ipmasqadm portfw -a -P tcp -L $IPADDR 2224 -R 192.168.1.x 2224
            ipmasqadm portfw -a -P tcp -L $IPADDR 21 -R 192.168.1.x 21
            ipmasqadm portfw -a -P tcp -L $IPADDR 20 -R 192.168.1.x 20
```

Testing and installation of the firewall

When we were ready to test, I created a psuedo network by connecting interface1 to a hub containing only the firewall and 1 test PC. The IP address of the 1st interface on the firewall was a CLASS C private IP of 192.168.1.10. I connected the 2nd interface to the outside network and assigned it a public IP address.

During testing, I had all ipmasq entries redirect to the PC. Then we (I enrolled outside colleagues to help me test the system. It's always nice to have an extra set of eyes to check your work.) tried coming into the firewall on each of the ports specified in the ipmasq list to make sure the firewall sent them to the PC. We enabled logging on all ACCEPT lines and checked the log files to make sure the firewall was handling the packets properly. We sent POP3 email to the PC and attempted DNS lookups *from* the PC. We also tested our rejects by trying incoming port 80 (http) and 53 (DNS) requests and running portscans using nmap against the firewall.

After all testing was complete and the firewall was working as intended, I replaced the Ascend router with the Cisco router, installed a mini-hub on the outside network. The mini-hub contains the Cisco router and the Linux firewall (outside interface). Then, we attached the firewall to the internal network and changed the IP address to match the old Ascend router address. This way, we didn't need to change all of the gateway/router settings on the PCs.

Installing an IDS

After the official "firewall installation" project was complete, I installed Snort IDS on an internal server to monitor the network for possible intrusions. We have a small IT staff, and we're unable to review the large logs generated on the outside network. A network-based IDS on the inside provides us with alerts of possible attacks and portscans, should someone get past (or around) our firewall. A script was created to send alerts to the network administrator.

After installing snort (see: http://www.snort.org/docs/ for lots of documentation), you'll need to make sure your network interface is running in promiscuous mode before starting up snort. To check your interface, run `'ifconfig eth0'`. You should see PROMISC under options. To run in promiscuous mode, type: `'ifconfig eth0 promisc'` and check again.

After starting snort, you can check /var/log/snort/alerts for alerts. I wrote a simple script to send new alerts to me via email.

You'll need to fine-tune the snort.conf file to meet your network's needs. A few of the false-positives we had to change on our configuration are:

- DHCP: our network uses dhcp rather than static IP addresses for our Windows workstations. These were generating lots of false positives.
- To avoid DNS false alarms from our ISP servers, I added the following line:

```
var DNS_SERVERS [x.x.x.x,x.x.x.x]
```

In addition to Snort IDS, I mentioned earlier that we run tcpwrappers on our servers. Each server contains a hosts.allow file with the specific clients/employees who are allowed access. Our tcpwrappers policy is "deny all except what is specifically allowed". Here is a sample:

Our hosts.deny file is set to deny all by default:

```
/etc $more /etc/hosts.deny
ALL:ALL
```

Our hosts.allow file has specific IP addresses that are allowed to access each server:

```
/etc $more /etc/hosts.allow
ALL: x.x.x.x
ALL: @ourdomain ALL@ALL
```

Interpreting Firewall Logs

The firewall will log all rules containing the '-l' option. Here is a typical firewall log entry:

```
Aug 13 11:14:34 hostname kernel: packet log: input DENY eth1 PROTO=17
10.x.x.x:137 217.x.x.x:137 L=78 s=0x00 I=2103 F=0x000 T=123 #115
```

The first few fields contain the date, time, and hostname. The packet information begins after the 'packet log:' field and represents:

- 'input' is the chain which contained the rule. In ipchains, the most common are 'input' and 'output'
- DENY is what the rule said to do with the packet. Valid ipchains options here are DENY, REJECT, ACCEPT, MASQ, REDIRECT, and RETURN.

- 'eth1' is the network interface
- 'PROTO=17' is the protocol (sample is UDP). Other common protocols include 6=TCP and 1=ICMP
- The first IP address/port combination is the packet's source address and port (IP=10.x.x.x port=137 in the example above). Port 137 is netbios Nameservice. You can see a complete listing of common ports by typing 'more /etc/services' at the Linux command line.
- The second IP address/port combination is the packet's destination address and port (IP=217.x.x.x port=137).
- The remaining fields aren't used much in log analysis, but represent the following:
    - L=78 is the packet's total length in bytes (includes header and data)
    - S=0x00 is the type of service (TOS) field
    - I=2103 is the datagram ID
    - F=0x000 is the fragment byte offset
    - T=123 is the time-to-live (TTL)
    - #115 is the ipchains rule number

Here are a few things to look out for when analyzing your firewall logs. (Note: firewall logs are not the only place to look for intruders or compromised systems, but is a good place to start.)

- If you have the firewall configured to log ACCEPTed packets of specific types, you can verify that these packets are what you intended to ACCEPT. For example, I have my firewall configured to log all SSH accepts. If the date and time are outside normal business hours, I question this access, since our business operates 8-5, Monday-Friday. Any incoming access outside of our business hours is questionable.
- Look for commonly probed ports. Most firewall books include information on this. You can also find a list of commonly probed ports at: http://www.linux-firewall-tools.com/linux/ports.html
- Look for a large jump in total logs in a given time period or from a specific IP or port range. This may indicate your firewall is being targeted for an attack.
- You may end up with thousands of logged packets per day. If so, you'll never be able to adequately analyze them manually. You'll need to use an automated log analysis package, which is discussed in the next section.

Log Analysis Packages

There are several good log analysis packages available for Linux. RedHat offers two packages in their standard installation, but there are plenty of others available. Here are the two provided by RedHat:

1. swatch - the 'simple watcher' program. Swatch can be run continually as a daemon, or by cron on a scheduled basis. For information on swatch, see: http://www.oit.ucsb.edu/~eta/swatch/swatch.html
2. logcheck – now called LogSentry. Logcheck/LogSentry automatically monitors your system logs and mails security violations to you. Most default installations of RedHat include logcheck. For more information, see: http://www.psionic.com/products/logsentry.html

Converting from ipchains to iptables

Since the ipchains project was completed, a newer version of Linux firewalling has been developed. The new version, called iptables, provides stateful filtering. Using the iptables tutorial by Oskar Andreasson (available at: http://people.unix-fu.org/andreasson/ipbles-tutorial/iptables-tutorial.html), I have successfully converted our ipchains firewall to iptables. Another good source of information is Rusty Russell's Packet Filtering HOWTO: http://netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/packet-filtering-HOWTO.linuxdoc.html. In this document, Rusty Russell explains the differences between ipchains and iptables. A sample iptables starting script is provided in appendix B.

To run iptables, there are two pieces of software that must be available. The kernel portion is called Netfilter, while the ruleset portion is referred to as iptables. The kernel requires several changes before you can use iptables:

- Your kernel should not run IPX or AppleTalk, which can be used to circumvent iptables rules.
- The fast-switching option must be set 'off' because fast-switching bypasses Netfilter completely.
- Enable 'code maturity-level options for development' in the kernel
- Under 'Networking Options', enable 'Network packet filtering (replaces ipchains)'. You don't need to enable Network packet-filtering debugging unless you have TONS of free space.
- Under 'IP: Netfilter Configuration, select all options as modules. The modules will be loaded as needed.

For more information on how to change and rebuild the kernel, refer to the HOWTO : www.tldp.org/HOWTO/Kernel-HOWTO.html

Since ipchains will *not* run with iptables, you'll need to shut off ipchains before iptables can be started. Do this with the following command:

```
chkconfig --level 0123456 ipchains off
```

You can stop the currently-running service by running the script under `/etc/rc.d/init.d/scriptname stop`, or by using:

```
service ipchains stop
```

There is an excellent article by David A. Bandel (LinuxJournal) on implementation of iptables available at: http://www.linuxjournal.com/article.php?sid=4815. The article covers how to check your Linux installation for all necessary modules, how to get the latest version of iptables, and how to use the iptables command line.

Ongoing Administration

An important part of any new firewall project includes ongoing administration and documentation of the security policy as it relates to the firewall, router, IDS, and user PCs. Security policies were created to define the remote access policy, firewall administration policy, and acceptable use policy. After we've gone through all of this effort to install a secure firewall, we need to make sure our PC users aren't installing dialup modems or using Gnutella to bypass the firewall entirely. As a sample, I've included a copy of our Remote Access Policy in Appendix A.

In addition to security policies, standard procedures were documented to provide instruction on the following:

- how to set up remote access
- how to properly administer the firewall
- how to configure a new PC for the network

Summary

The overall security of the business network was enhanced significantly with the addition of a packet-filtering ipchains firewall. This security was taken a step further with timely OS updates using up2date, the addition of tripwire software, and a Snort IDS. By converting ipchains to iptables, we've added stateful filtering to further enhance our security level. Finally, by documenting our security policies and providing clear instruction on the ongoing administration of the firewall, we are able to protect the network from inadvertent security holes caused by staff ignorance.

References

Web Sites:

Ward, Brian; Dev, Al. "Linux Kernel HOWTO"
URL: www.tldp.org/HOWTO/Kernel-HOWTO.html

RedHat Network
URL: http://rhn.redhat.com/network

Trustees of Indiana University, "Automating tasks with cron services"
URL: http://www.uwsg.iu.edu/usail/automation/cron.html

Russell, Rusty. "Linux ipchains HOWTO"
URL: http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html

Russell, Rusty. "Linux Packet Filtering HOWTO". 20 November 2001
URL:http://netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/packet-filtering-HOWTO.linuxdoc.html (10 April 2002)

Sebastien, Jean. "rcf script"
URL: http://rcf.mvlan.net:8080/

Ziegler, Robert L. "Commonly probed ports and linux firewall tools"
URL: http://www.linux-firewall-tools.com/linux

Atkins, E. Todd; Hansen, Stephen E., Stanford University "Swatch – the simple watcher program"
URL: http://www.oit.ucsb.edu/~eta/swatch/swatch.html

Psionic Technologies, "LogSentry (formerly Logcheck)"
http://www.psionic.com/products/logsentry.html

Andreasson, Oskar. "iptables tutorial 1.1.11"
URL: http://people.unix-fu.org/andreasson/iptables-tutorial/iptables-tutorial.html

Bandel, David A. "Taming the Wild Netfilter". LinuxJournal September 01, 2001
URL: http://www.linuxjournal.com/article.php?sid=4815

Books:

Ziegler, Robert L. Linux Firewalls New Riders Professional. November 1999

Zwicky, Elizabeth; Cooper, Simon; Chapman, D. Brent; Russell, Deborah. Building Internet Firewalls. O'Reilly & Associates. 15 January 2000

Appendix A

Remote Access Policy

1.0 Purpose

The purpose of this policy is to define standards for customers, employees, and consultants connecting to the corporate internal servers. These standards are designed to minimize the potential exposure to the Company from damages, which may result from unauthorized use of company resources. Damages include the loss of sensitive or company confidential data, intellectual property, damage to public image, and damage to company internal systems.

2.0 Scope

This policy applies to all remote access users, including but not limited to customers, consultants, and employees who require remote access into any of the company's internal servers.

3.0 Background

The Company allows their customers remote access into the application server for the purpose of entering orders and checking on inventory status. IT employees and consultants also require remote-access into company servers for support purposes.

4.0 Policy
4.1 General

Remote access into Company servers is provided using Secure Shell (ssh). Each server runs Secure Shell on a unique port. The firewall forwards approved incoming ssh requests to the proper server based on port number. All customer ssh users must be approved by their Company Sales Rep and all IT ssh users must be approved by the IT Manager for remote access. Upon approval, the Network Administrator establishes the remote access user accounts.

4.2 Responsibilities and Requirements

a. The Network Administrator will download and install the client copy of Secure Shell (ssh) for customer and employee use. Consultants must download and install their own copy of ssh client.
b. Ongoing administration of remote access users will be performed by the Network Administrator. Users will be prompted to change their passwords

every 90 days. The Network Administrator will monitor account activity and expire any accounts that are inactive for 90 days. If an account is needed after it has been expired, a request must be made in writing to the Network Administrator, who will reactivate the account. These procedures are outlined in detail in the "Administration of remote ssh users" document.

c. It is the responsibility of the remote access user to keep their login and password private. This information cannot be shared with anyone, at any time, for any reason.

d. Remote access to the Company servers must be used for business purposes only. Misuse of Company system resources is considered a violation of this policy.

e. All hosts that are connected to Company internal servers via remote access must use the most up-to-date anti-virus software.

5.0 Enforcement

Any employee found to have violated this policy may be subject to disciplinary action, up to and including termination.

6.0 Responsibility of this Policy

The IT Manager maintains this policy. It is the responsibility of the IT Manager to ensure that the policy is current, relevant, and enforced. This policy will be reviewed by the IT Manager on a quarterly basis.

7.0 Actions

IT Manager will review this policy every 3 months.

Remote-access users will change their passwords every 90 days.

8.0 Revision History

Original Policy written on April 23, 2002.

Appendix B – iptables script

```
#!/bin/sh
#
# rc.firewall - Initial SIMPLE IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson &lt;bluefluxATkoffeinDOTnet&gt;
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#
##############################################################################
#
# 1. Configuration options.
#
#
# 1.1 Internet Configuration.
#
INET_IP="xxx.xxx.xxx.xxx"
INET_IFACE="eth1"

# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP adress. the same as netmask 255.255.255.0
#

LAN_IP="192.168.1.1"
LAN_IP_RANGE="192.168.1.0/24"
LAN_BCAST_ADRESS="192.168.1.255"
LAN_IFACE="eth0"

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

##############################################################################
#
# 2. Module loading.
#
#
# Needed to initially load modules
#

/sbin/depmod -a
```

```
#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
/sbin/modprobe ipt_MASQUERADE
/sbin/modprobe ip_conntrack_ftp

########################################################################
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward
echo "30" > /proc/sys/net/ipv4/tcp_fin_timeout
echo "1800" > /proc/sys/net/ipv4/tcp_keepalive_intvl

########################################################################
#
# 4. rules set up.
#
#
# 4.1.1 Set default policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#
#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N icmp_packets
$IPTABLES -N tcp_packets
$IPTABLES -N udpincoming_packets

#
# 4.1.3 Create content in userspecified chains
#
#
```

```
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 2220 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 2221 -j allowed
# web
#$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
# identd
#$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

# if I'm a DNS server, uncomment
#$IPTABLES -A udpincoming_packets -p UDP -s 0/0 --destination-port 53 -j ACCEPT
# uncomment for NTP
#$IPTABLES -A udpincoming_packets -p UDP -s 0/0 --destination-port 123 -j ACCEPT
#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#
#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -d $LAN_BCAST_ADRESS -j ACCEPT

#
# Rules for incoming packets from the internet.
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udpincoming_packets
```

```
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#
#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

#
# 4.2.4 PREROUTING chain
#
# port 22
iptables -t nat -A PREROUTING -i $INET_IFACE -p tcp -s 0/0 -d $INET_IP \
--destination-port 22 -j DNAT --to-destination 192.168.1.x
# port 2220
iptables -t nat -A PREROUTING -i $INET_IFACE -p tcp -s 0/0 -d $INET_IP \
--destination-port 2220 -j DNAT --to-destination 192.168.1.x
# port 2221
iptables -t nat -A PREROUTING -i $INET_IFACE -p tcp -s 0/0 -d $INET_IP \
--destination-port 2221 -j DNAT --to-destination 192.168.1.x
# port 2223
```

```
iptables -t nat -A PREROUTING -i $INET_IFACE -p tcp -d $INET_IP\
--destination-port 2223 -j DNAT --to-destination 192.168.1.x
# port 2224
iptables –t nat –A PREROUTING –I $INET_IFACE –p tcp –s 0/0 –d $INET_IP \
--destination-port 2224 –j DNAT --to-destination 192.168.1.x


#
# 4.2.5 POSTROUTING chain
#
#
# Enable simple IP Forwarding and Network Address Translation
#
# this does the entire internal network
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source $INET_IP
```