Firewalling and Proxy Server HOWTO

Mark Grennan, markg@netplus.net v0.4, 8 November 1996

This document is designed to teach the basics of firewall systems and give you some detail on setting up both a filtering and proxy firewall on a Linux based PC. An HTML version of this document is available at *http://okcforum.org/~markg/Firewall-HOWTO.html*

Contents

1 Introduction **2**

1.1 Feedback . . . 3

1.2 Disclaimer . . . 3

1.3 Copyright . . . 3

1.4 My Reasons for Writing This . . . 3

1.5 TODO . . . 4

1.6 Further Readings . . . 4

2 Understanding Firewalls **4**

2.1 Drawbacks with Firewalls . . . 5

2.2 Types of Firewalls . . . 5

2.2.1 IP Filtering Firewalls . . . 5

2.2.2 Proxy Servers . . . 5

3 Setting up the Firewall **6**

3.1 Hardware requirements . . . 6

4 Firewalling Software **6**

4.1 Available packages . . . 6

4.2 The TIS Firewall Toolkit vs SOCKS . . . 6

5 Preparing the Linux system **7**

5.1 Compiling the Kernel . . . 7

5.2 Configuring two network cards . . . 8

5.3 Configuring the Network Addresses . . . 8

5.4 Testing your network . . . 9

5.5 Securing the Firewall 10

6 IP filtering setup (IPFWADM) 11

7 Installing the TIS Proxy server 12

7.1 Getting the software 12

7.2 Compiling the TIS FWTK 13

7.3 Installing the TIS FWTK 14

7.4 Configuring the TIS FWTK 14

7.4.1 The netperm-table file 15

7.4.2 The inetd.conf file 18

7.4.3 The /etc/services file 20

8 The SOCKS Proxy Server 20

8.1 Setting up the Proxy Server 20

8.2 Configuring the Proxy Server 21

8.2.1 The Access File 21

8.2.2 The Routing File 22

8.3 Working With a Proxy Server 23

8.3.1 Unix 23

8.3.2 MS Windows with Trumpet Winsock 23

8.3.3 Getting the Proxy Server to work with UDP Packets 23

8.4 Drawbacks with Proxy Servers 23

9 Advanced Configurations 24

9.1 A large network with emphasis on security 24

9.1.1 The Network Setup 25

9.1.2 The Proxy Setup 25

1 Introduction

This original Firewall-HOWTO was written by David Rudder, **drig@execpc.com**. I'd like to thank him for allowing me to update his work.

Firewalls have gained great fame recently as the ultimate in Internet Security. Like most things that gain fame, with that fame has come misunderstanding. This HOWTO will go over the basics of what a firewall

is, how to set one up, what proxy servers are, how to set up proxy servers, and the applications of this technology outside of the security realm.

1.1 Feedback

Any feedback is very welcome. **PLEASE REPORT ANY INACCURACIES IN THIS PAPER!!!** I am human, and prone to making mistakes. If you find any, fixing them is of my highest interest. I will try to answer all e-mail, but I am busy, so don't get insulted if I don't.

My email address is **markg@netplus.net**

1.2 Disclaimer

I AM NOT RESPONSIBLE FOR ANY DAMAGES INCURRED DUE TO ACTIONS TAKEN BASED ON THIS DOCUMENT. This document is meant as an introduction to how firewalls and proxy servers work. I am not, nor do I pretend to be, a security expert. I am just some guy who has read to much and likes computers more than most people. Please, I am writing this to help get people acquainted with this subject, and I am not ready to stake my life on the accuracy of what is in here.

1.3 Copyright

Unless otherwise stated, Linux HOWTO documents are copyrighted by their respective authors. Linux HOWTO documents may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained on all copies. Commercial redistribution is allowed and encouraged; however, the author would like to be notified of any such distributions.

All translations, derivative works, or aggregate works incorporating any Linux HOWTO documents must be covered under this copyright notice. That is, you may not produce a derivative work from a HOWTO and impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the Linux HOWTO coordinator.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the HOWTO documents, and would like to be notified of any plans to redistribute the HOWTOs.

If you have any questions, please contact Mark Grennan at <markg@netplus.net>.

1.4 My Reasons for Writing This

Even though there were a lot of discussions on comp.os.linux.* over the past year about firewalling, I found it difficult to find the information I needed to setup a firewall. The original version of this HOWTO was helpful but still lacking. I hope this beefed up version of David Rudder's Firewall HOWTO will give everyone the information they need to create a functioning firewall in hours, not weeks.

I also feel I should return something to the Linux community.

1.5 TODO

- Give some instructions on how to setup the clients
- Find a good UDP proxy server that works with Linux

1.6 Further Readings

- The NET-2 HOWTO
- The Ethernet HOWTO
- The Multiple Ethernet Mini HOWTO
- Networking with Linux
- The PPP HOWTO
- TCP/IP Network Administrator's Guide by O'Reilly and Associates
- The Documentation for the TIS Firewall Toolkit

Trusted Information System's (TIS) web site has a great collection of documentation on firewalls and related meterial. **http://www.tis.com/**

Also, I am working on a security project called I am calling *Secure Linux*. On the *Secure Linux* web site I am gathering all the information, documemtation and programs you need to create a trusted Linux system. Email me if you would like information.

2 Understanding Firewalls

A firewall is a term used for a part of a car. In cars, firewalls are physical objects that separate the engine from the passengers. They are meant to protect the passenger in case the car's engine catches fire while still providing the driver access to the engine's controls.

A firewall in computers is a device that protects a private network from the public part (the internet as a whole).

The firewall computer, from now on named "firewall", can reach both the protected network and the internet. The protected network can't reach the internet, and the internet can not reach the protected network.

For someone to reach the internet from inside the protected network, they must telnet to firewall, and use the internet from there.

The simplest form of a firewall is a dual homed system. (a system with two network connections) If you can TRUST ALL your users, you can simple setup a Linux (compile it with IP forwarding/gatewaying turned OFF!) and give everyone accounts on it. The can then login to this system and telnet, FTP, read mail, and use any other service you provided. With this setup, the only computer on your private network

that knows anything about the outside world is the firewall. The other system on your protected network dont even need a default route.

This needs re-stating. For the above firewall to work **YOU MUST TRUST ALL YOUR USERS!** I don't recommend it.

2.1 Drawbacks with Firewalls

The problem with filtering firewalls are they inhibit the access to your network from the internet. Only services on systems that have pass filters can be accessed. With a proxy server users can login to the firewall and then access any system within the private network they have access to.

Also, new types of network clients and servers a coming out almost daily. When they do you must find a new way to allow controled access before these services can be used.

2.2 Types of Firewalls

There are two types of firewalls.

1. IP or Filtering Firewalls - that block all but selected network traffic.
2. Proxy Servers - that make the network connections for you.

2.2.1 IP Filtering Firewalls

An IP filtering firewall works at the packet level. It is designed to control the flow of packets based the source, destination, port and packet type information contained in each packet.

This type of firewall is very secure but lacks any sort of useful logging. It can block people from accessing private system but it will not tell you who accessed your public systems or who accessed the internet from the inside.

Filtering firewalls are absolute filters. Even if you want to give someone on outside access to your private servers you can not without giving everyone access to the servers.

Linux has included packet filtering software in the kernel starting with version 1.3.x.

2.2.2 Proxy Servers

Proxy servers allow indirect internet access through the firewall. The best example of how this works is a person telneting to a system and then telneting from there to another. Only with a proxy server the process is automatic. When you connect to a proxy server with your client software, the proxy server starts it's client (proxy) software and passes you the data.

Because proxy servers are duplicating all the communications they can log every thing they do.

The great thing about proxy servers is that they are completely secure, when configured correctly. They will not allow someone in through them. There are no direct IP routes.

3 Setting up the Firewall

3.1 Hardware requirements

For our example, the computer is a 486-DX66 with 16 meg of memory and a 500 meg Linux partition. This system has two network cards one connected to our private LAN and the other connected to the a lan we will call the de-militarized zone (DMZ). The DMZ has a router connected to it with a connection to the internet.

This is a pretty standard setup for a business. You could use one network card and a modem with PPP to the internet. The point is, the firewall must have two IP network numbers.

I know a lot of people have small LANs at home with two or three computers on them. Something you might consider is putting all your modems in on Linux box (maybe an old 386) and connecting all of them to the internet with load balancing. With this setup when only one person was pulling data they would get both modems doubling the throughput. :-)

4 Firewalling Software

4.1 Available packages

If all you want is a filtering firewall, you only need Linux and the basic networking packages. One package that might not come with your distribution is the IP Firewall Administration tool.

(IPFWADM) Comes from **http://www.xos.nl/linux/ipfwadm/**

If you want to setup a poxy server you will need one of these packages.

1. SOCKS
2. TIS Firewall Toolkit (FWTK)

4.2 The TIS Firewall Toolkit vs SOCKS

Trusted Information System (**http://www.tis.com**) has put out a collection of programs designed to facilitate firewalling. The programs do basically the same thing as the SOCKS package, but with a different design strategy. Where Socks has one program that covers all Internet transactions, TIS has provided one program for each utility that wishes to use the firewall.

To contrast the two, let's use the example of world wide web and Telnet access. With SOCKS, you set up one configuration file and one daemon. Through this file and daemon, both telnet and WWW are enabled, as well as any other service that you have not disabled.

With the TIS toolkit, you set up one daemon for each WWW and telnet, as well as configuration files for each. After you have done this, other internet access is still prohibited until explicitly set up. If a daemon for a specific utility has not been provided (like talk), there is a "plug-in" daemon, but it is neither as flexible, nor as easy to set up, as the other tools.

This might seem a minor, but it makes a major difference. SOCKS allows you to be sloppy. With a poorly set up SOCKS server, someone from the inside could gain more access to the internet than was originally intended. With the TIS toolkit, the people on the inside have only the access the system administrator wants them to have.

SOCKS is easier to set up, easier to compile and allows for greater flexibility. The TIS toolkit is more secure if you want to regulate the users inside the protected network. Both provide absolute protection from the outside.

I will cover the installation and setup of both.

5 Preparing the Linux system

5.1 Compiling the Kernel

Start with a clean installation of your Linux distribution. (I used RedHat 3.0.3 and the examples here are based on this distribution.) The less software you have loaded the less holes, backdoors and/or bugs there will be to introduce security problems in your system, so load only a minimal set of applications.

Pick a stable kernel. I used the Linux 2.0.14 kernel for my system. So this documentation is based on it's settings.

You well need to recompile the Linux kernel with the appropriate options. At this point, you should look at the Kernel HOWTO, the Ethernet HOWTO, and the NET-2 HOWTO if you haven't done this before.

Here are the network related setting I know work in 'make config'

1. Under General setup
   (a) Turn Networking Support ON
2. Under Networking Options
   (a) Turn Network firewalls ON
   (b) Turn TCP/IP Networking ON
   (c) Turn IP forwarding/gatewaying OFF (UNLESS you wish to use IP filtering)
   (d) Turn IP Firewalling ON
   (e) Turn IP firewall packet loggin ON (this is not required but it is a good idea)
   (f) Turn IP: masquerading OFF (I am not covering this subject here.)
   (g) Turn IP: accounting ON
   (h) Turn IP: tunneling OFF
   (i) Turn IP: aliasing OFF
   (j) Turn IP: PC/TCP compatibility mode OFF
   (k) Turn IP: Reverse ARP OFF
   (l) Turn Drop source routed frames ON

3. Under Network device support
   (a) Turn Network device support ON
   (b) Turn Dummy net driver support ON
   (c) Turn Ethernet (10 or 100Mbit) ON
   (d) Select your network card

Now you can recompile, reinstall the kernel and reboot. Your network card/s should show up in the boot-up sequence. If not, go over the other HOWTOs again until it is working.

5.2 Configuring two network cards

If you have two network cards in your computer, you most likely will need to add an append statement to your /etc/lilo.conf file to describe the IRQ and address of both cards. My lilo append statement looks like this:

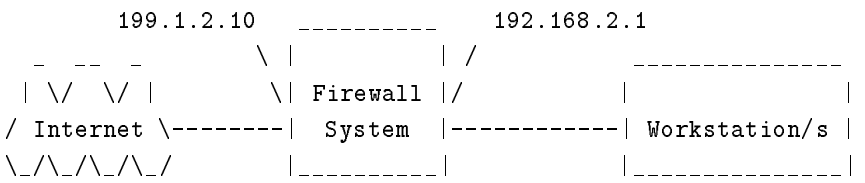
```
append={\tt "}ether=12,0x300,eth0 ether=15,0x340,eth1{\tt "}
```

5.3 Configuring the Network Addresses

This is the real interesting part. Now you have a few decisions to make. Since we don't want the internet to have access to any part of the private network, we do not need to use real addresses. There are a number of internet addresses set aside for private networks. Because everyone needs more addresses and because these addresses can not cross the Internet they are a good choice.

Of these, 192.168.2.xxx, is set aside and we will use it in our examples.

Your proxy firewall will be a member of both networks and so it can pass the data through to and from the private network.



If your going to use a filtering firewall you can still use these numbers. You will need to use IP masquerading to make this happen. With this process the firewall will forward packets and translate them into "REAL " " IP address to travel on the Internet.

You must assign the real IP address to the network card on the Internet (out) side. And, assign 192.168.2.1 to the Ethernet card on inside. This will be your proxy/gateway IP address. You can assign all the other machines in the protected network some number in that 192.168.2.xxx range. (192.168.2.2 through 192.168.2.254)

Since I use RedHat Linux (Hey guys, want to give me a copy for the plugs? ;-) to configure the network at boot time I added a 'ifcfg-eth1' file in the /etc/sysconfig/network-scripts directory. This file is read during the boot process to set your network and routing tables.

Here is what my ifcfg-eth1 looks like;

```
#!/bin/sh
#>>>Device type: ethernet
#>>>Variable declarations:
DEVICE=eth1
IPADDR=192.168.2.1
NETMASK=255.255.255.0
NETWORK=192.168.2.0
BROADCAST=192.168.2.255
GATEWAY=199.1.2.10
ONBOOT=yes
#>>>End variable declarations
```

You can also use these scripts to automatically connect by modem to your provider. Look at the ipup-ppp script.

If your going to use a modem for your internet connection your outside IP address will be assigned for you by your provider at connect time.

5.4 Testing your network

Start by checking ifconfig and route. If you have two network cards your ifconfig should look something like:

```
#ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.0  Bcast:127.255.255.255  Mask:255.0.0.0
          UP BROADCAST LOOPBACK RUNNING  MTU:3584  Metric:1
          RX packets:1620 errors:0 dropped:0 overruns:0
          TX packets:1620 errors:0 dropped:0 overruns:0

eth0      Link encap:10Mbps Ethernet  HWaddr 00:00:09:85:AC:55
          inet addr:199.1.2.10 Bcast:199.1.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
          Interrupt:12 Base address:0x310

eth1      Link encap:10Mbps Ethernet  HWaddr 00:00:09:80:1E:D7
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
          Interrupt:15 Base address:0x350
```

and your route table sould look like:

```
#route -n
Kernel routing table
Destination     Gateway         Genmask         Flags MSS    Window Use Iface
199.1.2.0       *               255.255.255.0   U     1500   0       15 eth0
192.168.2.0     *               255.255.255.0   U     1500   0        0 eth1
127.0.0.0       *               255.0.0.0       U     3584   0        2 lo
default         199.1.2.10      *               UG    1500   0       72 eth0
```

Note: 199.1.2.0 is the Internet side of this firewall and 192.168.2.0 is the private side.

Now try to ping the internet from the firewall. I used to use nic.ddn.mil as my test point. It's still a good test, but has proven to be less reliable than I had hoped. If it doesn't work at first, try pinging a couple other places that are not connected to your LAN. If this doesn't work, then your PPP is incorrectly setup. Reread the Net-2 HOWTO, and try again.

Next, try pinging a host within the protected network from the firewall. All the computers should be able to ping each other. If not, go over the NET-2 HOWTO again and work on the network some more.

Then, try to ping the outside address of firewall from inside the protected network. (NOTE: this is not any of the 192.168.2.xxx IP numbers.) If you can, then you have not turned off IP Forwarding. Make sure this is the way you want it. If you leave it turned on you will have to go through the IP filtering section of this document as well.

Now try pinging the internet from behind your firewall. Use the same address that worked for you before. (I.E. nic.ddn.mil) Again, if you have IP Forwarding turned off, this should not work. But, if you have it turned on, it should.

If have IP Forwarding turned on and your using a "REAL" (not 192.168.2.*) IP address for your private network, and you can't ping the internet but you can ping the internet side your firewall, check if the next router up stream is routing packets for your private network address. (Your provider may have to do this for you.)

If you have assigned your protected network to 192.168.2.*, then no can packets can be routed to it anyway. If you have skipped ahead and you already have IP masquerading turn on, this test should work.

Now, you have your basic system setup.

5.5 Securing the Firewall

A firewall isn't any good if it is left wide open to attacks through a unused service. A "bad guy" could gain access to the firewall and modify it for their own needs.

Start by turning off any unneeded services. Look at /etc/inetd.conf file. This file controls what are called the "super server". It controls a bunch of the server daemons and starts them as they are requested.

Definitely turn off netstat, systat, tftp, bootp, and finger. To turn a service off, put # as the first character of the service line. When your done, send a SIG-HUP to the process by typing **"kill -HUP <pid>"**, where <pid> is the process number of inetd. This will make inetd re-read its configuration file (inetd.conf) and restart.

Test it out by telneting to port 15 on firewall, the netstat port. If you get an output of netstat, you have not restarted it correctly.

6 IP filtering setup (IPFWADM)

To start, you should have IP Forwarding turned on in your kernel and your system should be up and forwarding everything you send it. Your routing tables should be in place and you should be able to access everything, both from the inside out and from the outside in.

But, we're building a firewall so we need to start chocking down what everyone has access to.

In my system I created a couple of scripts to set the firewall forwarding policy and accounting policy. I call theses scripts from the /etc/rc.d scripts so my system is configured at boot time.

By default the IP Forwarding system in the Linux kernel forwards everything. Because of this, your firewall script should start by denying access to everything and flushing any ipfw rules in place from the last time it was run. This script will do the trick.

```
#
# setup IP packet Accounting and Forwarding
#
#   Forwarding
#
# By default DENY all services
ipfwadm -F -p deny
# Flush all commands
ipfwadm -F -f
ipfwadm -I -f
ipfwadm -O -f
```

Now we have the ultimate firewall. Nothing can get through. No doubt you have some services you need to forward so here are a few examples you should find useful.

```
# Forward email to your server
ipfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 1024:65535 -D 192.1.2.10 25

# Forward email connections to outside email servers
ipfwadm -F -a accept -b -P tcp -S 196.1.2.10 25 -D 0.0.0.0/0 1024:65535
```

```
# Forward Web connections to your Web Server
/sbin/ipfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 1024:65535 -D 196.1.2.11 80

# Forward Web connections to outside Web Server
/sbin/ipfwadm -F -a accept -b -P tcp -S 196.1.2.* 80 -D 0.0.0.0/0 1024:65535

# Forward DNS traffic
/sbin/ipfwadm -F -a accept -b -P udp -S 0.0.0.0/0 53 -D 196.1.2.0/24
```

You might also be interested in accounting for traffic going through your firewall. This script will count ever packet. You could add a line or to to account for packets going to just a single system.

```
# Flush the current accounting rules
ipfwadm -A -f
# Accounting
/sbin/ipfwadm -A -f
/sbin/ipfwadm -A out -i -S 196.1.2.0/24 -D 0.0.0.0/0
/sbin/ipfwadm -A out -i -S 0.0.0.0/0 -D 196.1.2.0/24
/sbin/ipfwadm -A in -i -S 196.1.2.0/24 -D 0.0.0.0/0
/sbin/ipfwadm -A in -i -S 0.0.0.0/0 -D 196.1.2.0/24
```

If all you wanted was a filtering firewall you can stop here. Enjoy :-)

7 Installing the TIS Proxy server

7.1 Getting the software

The TIS FWTK is avaible at **ftp://ftp.tis.com/**.

Don't make the mistake I did. When you ftp files from TIS, READ THE README's. The TIS fwtk is locked up in a hidden directory on their server. TIS requires you **send email to fwtk-request@tis.com** with only the word **SEND** in the body of the message to learn the name of this hidden directory. No subject is needed in the message. Their system will then mails you back the directory name (good for 12 hours) to download the source.

As I'm writing this TIS is releasing version 2.0 (beta) of the FWTK. This version seems to compile well (with a few exceptions) and everything is working for me. This is the version I will be covering here. When they release the final code I'll update the HOWTO.

To install the FWTK, create a fwtk-2.0 directory in your /usr/src directory. Move your copy of the FWTK (fwtk-2.0.tar.gz) to your this directory and untar it (tar zxf fwtk-2.0.tar.gz).

The FWTK does not proxy SSL web documents but there is an addon for it written by Jean-Christophe Touvet. It is avaible at **ftp://ftp.edelweb.fr/pub/contrib/fwtk/ssl-gw.tar.Z**. Touvet does not support this code.

I am using a modified version that includes access to Netscape secure news servers written by Eric Wedel. It is available at **ftp://mdi.meridian-data.com/pub/tis.fwtk/ssl-gw/ssl-gw2.tar.Z**.

In our example I will use Eric Wedel's version.

To install it, simply create a ssl-gw directory in your /usr/src/fwtk-2.0 directory and put the files in it.

When I installed this gateway it required a few changes before it would compile with the rest of the toolkit.

The first change was to the ssl-gw.c file. I found it didn't include a needed include file.

```
#if defined(__linux)
#include        <sys/ioctl.h>
#endif
```

Second it didn't come with a Makefile. I copied one out of the other gateway directories and replaced the gateway's name with ssl-gw.

7.2 Compiling the TIS FWTK

Version 2.0 of the FWTK compiles much easier then any of the older versions. I still found a couple of things that needed to be changed before the BETA version would compile cleanly. Hopefully these changes will be make in the final version.

To fix it up, start by changing to the /usr/src/fwtk/fwtk directory and coping the Makefile.config.linux file over the Makefile.config file.

DON'T RUN FIXMAKE. The instructions tell you to run this. If you do it will break the makefiles in each directory.

I do have a fix for fixmake. The problem is the sed script add a '.' and " to the include line of ever Makefile. This sed script works.

```
sed 's/^include[        ]*\([^  ].*\)/include \1/' $name .proto > $name
```

Next we need to edit the Makefile.config file. There are two changes you may need to make.

The author set the source directory to his home directory. We are compiling our code in /usr/src so you should changed the FWTKSRCDIR variable to reflect this.

```
FWTKSRCDIR=/usr/src/fwtk/fwtk
```

Second, at least some Linux system us the gdbm database. The Makefile.config is using dbm. You might need to change this. I had to for RedHat 3.0.3.

```
DBMLIB=-lgdbm
```

The last fix is in the x-gw. The bug in the BETA version is in the socket.c code. To fix it remove these lines of code.

```
#ifdef SCM_RIGHTS  /* 4.3BSD Reno and later */
                     + sizeof(un_name->sun_len) + 1
#endif
```

If you added the ssl-gw to your FWTK source directory you will need to add it to the list of directory in the Makefile.

```
DIRS=   smap smapd netacl plug-gw ftp-gw tn-gw rlogin-gw http-gw x-gw ssl-gw
```

Now run **make**.

7.3 Installing the TIS FWTK

Run **make install**.

The default installation directory is /usr/local/etc. You could change this (I didn't) to a more secure directory. I chose to change the access to this directory to 'chmod 700'.

All last is left now is to configure the firewall.

7.4 Configuring the TIS FWTK

Now the fun realy begins. We must teach the system to call theses new services and create the tables to control them.

I'm not going to try to re-write the TIS FWTK manual here. I will show you the setting I found worked and explain the problems I ran into and how I got around them.

There are three files that make up these controls.

- /etc/services
  - Tells the system what ports a services is on.
- /etc/inetd.conf
  - Tells inetd what program to call when someone knocks on a service port.
- /usr/local/etc/netperm-table
  - Tells the FWTK services who to allow and deny service to.

To get the FWTK functioning, you should edit these files from the bottom up. Editing the services file without the inetd.conf or netperm-table file set correctly could make your system inaccessible.

7.4.1 The netperm-table file

This file controls who can access the services of the TIS FWTK. You should think about the traffic using the firewall from both sides. People outside your network should identify themselves before gaining access, but the people inside your network might be allowed to just pass through.

So people can identify themselves, the firewall uses a program called **authsrv** to keep a database of user IDs and passwords. The authentication section of the netperm-table controls where the database is keep and who can access it.

I had some trouble closing the access to this service. Note the premit-hosts line I show uses a '*' to give everyone access. The correct setting for this line is " `authsrv: premit-hosts localhost` if you can get it working.

```
#
# Proxy configuration table
#
# Authentication server and client rules
authsrv:      database /usr/local/etc/fw-authdb
authsrv:      permit-hosts *
authsrv:      badsleep 1200
authsrv:      nobogus true
# Client Applications using the Authentication server
*:            authserver 127.0.0.1 114
```

To initialize the database, su to root, and run **./authsrv** in the /var/local/etc directory to create the administrative user record. Here is a sample session.

Read the FWTK documentation to learn how to add users and groups.

```
#
# authsrv
authsrv# list
authsrv# adduser admin {\tt "}Auth DB admin{\tt "}
ok - user added initially disabled
authsrv# ena admin
enabled
authsrv# proto admin pass
changed
authsrv# pass admin {\tt "}plugh{\tt "}
Password changed.
authsrv# superwiz admin
set wizard
authsrv# list
Report for users in database
user   group  longname          ok?    proto   last
```

```
------ ------ ------------------ -----  ------  -----
admin         Auth DB admin      ena    passw   never
authsrv# display admin
Report for user admin (Auth DB admin)
Authentication protocol: password
Flags: WIZARD
authsrv# ^D
EOT
#
```

The telnet gateway (tn-gw) controls are straight forward and the first you should set up.

In my example, I premit host from inside the private network to pass through without authenticating themselves. (permit-hosts 19961.2.* -passok) But, any other user must enter their user ID and password to use the proxy. (permit-hosts * -auth)

I also allow one other system (196.1.2.202) to access the firewall directly without going through the firewall at all. The two inetacl-in.telnetd lines do this. I will explain how these lines are called latter.

The Telnet timeout should be keep short.

```
# telnet gateway rules:
tn-gw:                denial-msg      /usr/local/etc/tn-deny.txt
tn-gw:                welcome-msg     /usr/local/etc/tn-welcome.txt
tn-gw:                help-msg        /usr/local/etc/tn-help.txt
tn-gw:                timeout 90
tn-gw:                permit-hosts 196.1.2.* -passok -xok
tn-gw:                permit-hosts * -auth
# Only the Administrator can telnet directly to the Firewall via Port 24
netacl-in.telnetd: permit-hosts 196.1.2.202 -exec /usr/sbin/in.telnetd
```

The r-commands work the same way as telnet.

```
# rlogin gateway rules:
rlogin-gw:    denial-msg      /usr/local/etc/rlogin-deny.txt
rlogin-gw:    welcome-msg     /usr/local/etc/rlogin-welcome.txt
rlogin-gw:    help-msg        /usr/local/etc/rlogin-help.txt
rlogin-gw:    timeout 90
rlogin-gw:    permit-hosts 196.1.2.* -passok -xok
rlogin-gw:    permit-hosts * -auth -xok
# Only the Administrator can telnet directly to the Firewall via Port
netacl-rlogind: permit-hosts 196.1.2.202 -exec /usr/libexec/rlogind -a
```

You shouldn't have anyone accessing your firewall directly and that includes FTP so don't put an FTP, server on you firewall.

Again, the permit-hosts line allows anyone in the protected network free access to the Internet and all others must authenticate themselves. I included logging of every file sent and received to my controls. (-log { retr stor })

The ftp timeout controls how long it will take to drop a bad connections as well as how long a connection will stay open with out activity.

```
# ftp gateway rules:
ftp-gw:               denial-msg      /usr/local/etc/ftp-deny.txt
ftp-gw:               welcome-msg     /usr/local/etc/ftp-welcome.txt
ftp-gw:               help-msg        /usr/local/etc/ftp-help.txt
ftp-gw:               timeout 300
ftp-gw:               permit-hosts 196.1.2.* -log { retr stor }
ftp-gw:               permit-hosts * -authall -log { retr stor }
```

Web, gopher and browser based ftp are contorted by the http-gw. The first two lines create a directory to store ftp and web documents as they are passing through the firewall. I make these files owned by root and put the in a directory accessible only by root.

The Web connection should be kept short. It controls how long the user will wait on a bad connections.

```
# www and gopher gateway rules:
http-gw:      userid          root
http-gw:      directory       /jail
http-gw:      timeout 90
http-gw:      default-httpd   www.afs.net
http-gw:      hosts           196.1.2.* -log { read write ftp }
http-gw:      deny-hosts      *
```

The ssl-gw is really just a pass anything gateway. Be carefull with it. In this example I allow anyone inside the protected network to connect to any server outside the network except the addresses 127.0.0.* and 192.1.1.* and then only on ports 443 through 563. Ports 443 through 563 are known SSL ports.

```
# ssl gateway rules:
ssl-gw:         timeout 300
ssl-gw:         hosts           196.1.2.* -dest { !127.0.0.* !192.1.1.* *:443:563 }
ssl-gw:         deny-hosts      *
```

Here is an example of how to use the plug-gw to allow connections to a news server. In this example I allow anyone inside the protected network to connect to only one system and only to it's news port.

The seconded line allows the news server to pass its data back to the protected network.

Because most clients expect to stay connected while the user read news, the timeout for a news server should be long.

```
# NetNews Pluged gateway
plug-gw:        timeout 3600
plug-gw: port nntp 196.1.2.* -plug-to 199.5.175.22 -port nntp
plug-gw: port nntp 199.5.175.22 -plug-to 196.1.2.* -port nntp
```

The finger gateway is simple. Anyone inside the protected network must login first and then we allow them to use the finger program on the firewall. Anyone else just gets a message.

```
# Enable finger service
netacl-fingerd: permit-hosts 196.1.2.* -exec /usr/libexec/fingerd
netacl-fingerd: permit-hosts * -exec /bin/cat /usr/local/etc/finger.txt
```

I haven't setup the Mail and X-windows services so I'm not including examples. If anyone has a working example, please send me email.

7.4.2 The inetd.conf file

Here is a complete /etc/inetd.conf file. All un-needed services have been commented out. I have included the complete file to show what to turn off, as well as how to setup the new firewall services.

```
#echo stream  tcp  nowait  root       internal
#echo dgram   udp  wait    root       internal
#discard      stream  tcp  nowait  root       internal
#discard      dgram   udp  wait    root       internal
#daytime      stream  tcp  nowait  root       internal
#daytime      dgram   udp  wait    root       internal
#chargen      stream  tcp  nowait  root       internal
#chargen      dgram   udp  wait    root       internal
# FTP firewall gateway
ftp-gw      stream  tcp  nowait.400  root  /usr/local/etc/ftp-gw  ftp-gw
# Telnet firewall gateway
telnet        stream  tcp  nowait      root  /usr/local/etc/tn-gw /usr/local/etc/tn-gw
# local telnet services
telnet-a    stream  tcp  nowait      root  /usr/local/etc/netacl in.telnetd
# Gopher firewall gateway
gopher        stream  tcp  nowait.400  root  /usr/local/etc/http-gw /usr/local/etc/http-gw
# WWW firewall gateway
http  stream  tcp  nowait.400  root  /usr/local/etc/http-gw /usr/local/etc/http-gw
# SSL firewall gateway
ssl-gw  stream  tcp     nowait  root /usr/local/etc/ssl-gw   ssl-gw
# NetNews firewall proxy (using plug-gw)
nntp    stream  tcp     nowait  root    /usr/local/etc/plug-gw plug-gw nntp
```

```
#nntp stream  tcp     nowait  root    /usr/sbin/tcpd  in.nntpd
# SMTP (email) firewall gateway
#smtp stream  tcp     nowait  root    /usr/local/etc/smap smap
#
# Shell, login, exec and talk are BSD protocols.
#
#shell        stream  tcp     nowait  root    /usr/sbin/tcpd  in.rshd
#login        stream  tcp     nowait  root    /usr/sbin/tcpd  in.rlogind
#exec stream  tcp     nowait  root    /usr/sbin/tcpd  in.rexecd
#talk dgram   udp     wait    root    /usr/sbin/tcpd  in.talkd
#ntalk        dgram   udp     wait    root    /usr/sbin/tcpd  in.ntalkd
#dtalk        stream  tcp     waut    nobody  /usr/sbin/tcpd  in.dtalkd
#
# Pop and imap mail services et al
#
#pop-2   stream  tcp  nowait  root  /usr/sbin/tcpd    ipop2d
#pop-3   stream  tcp  nowait  root  /usr/sbin/tcpd    ipop3d
#imap    stream  tcp  nowait  root  /usr/sbin/tcpd    imapd
#
# The Internet UUCP service.
#
#uucp    stream  tcp  nowait  uucp  /usr/sbin/tcpd  /usr/lib/uucp/uucico -l
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as {\tt "}boot servers.{\tt "} Do not uncomment
# this unless you *need* it.
#
#tftp dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
#bootps       dgram   udp     wait    root    /usr/sbin/tcpd  bootpd
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
#
# cfinger is for GNU finger, which is currently not in use in RHS Linux
#
finger       stream  tcp  nowait  root   /usr/sbin/tcpd  in.fingerd
#cfinger     stream  tcp  nowait  root   /usr/sbin/tcpd  in.cfingerd
#systat      stream  tcp  nowait  guest  /usr/sbin/tcpd  /bin/ps -auwwx
#netstat     stream  tcp  nowait  guest  /usr/sbin/tcpd  /bin/netstat -f inet
#
# Time service is used for clock syncronization.
#
#time stream  tcp  nowait  root  /usr/sbin/tcpd  in.timed
```

```
#time dgram   udp  wait    root  /usr/sbin/tcpd  in.timed
#
# Authentication
#
auth          stream  tcp  wait    root  /usr/sbin/tcpd  in.identd -w -t120
authsrv       stream  tcp  nowait  root  /usr/local/etc/authsrv authsrv
#
# End of inetd.conf
```

7.4.3 The /etc/services file

This is where it all begins. When a client connects to the firewall it connects on a known port (less then 1024). For example telnet connects on port 23. The inetd deamon hears this connection and looks up the name of these service in the /etc/services file. It then calls the program assigned to the name in the /etc/inetd.conf file.

Some of the services we are creating are not normally in the /etc/services file. You can assign some of them to any port you want. For example, I have assigned the administrator's telnet port (telnet-a) to port 24. You could assign it to port 2323 if you wished. For the administrator (YOU) to connect directly to the firewall you will need to telnet to port 24 not 23 and if you setup your netperm-table file, like I did, you will only be able to to this from one system inside your protected network.

```
telnet-a        24/tcp
ftp-gw          21/tcp           # this named changed
auth            113/tcp   ident    # User Verification
ssl-gw          443/tcp
```

8 The SOCKS Proxy Server

8.1 Setting up the Proxy Server

The SOCKS proxy server available from
ftp://sunsite.unc.edu/pub/Linux/system/Network/misc/socks-linux- src.tgz. There is also an example config file in that directory called "socks-conf". Uncompress and untar the files into a directory on your system, and follow the instructions on how to make it. I had a couple problems when I made it. Make sure that your Makefiles are correct.

One important thing to note is that the proxy server needs to be added to /etc/inetd.conf. You must add a line:

```
socks  stream  tcp  nowait  nobody  /usr/local/etc/sockd  sockd
```

to tell the server to run when requested.

8.2 Configuring the Proxy Server

The SOCKS program needs two separate configuration files. One to tell the access allowed, and one to route the requests to the appropriate proxy server. The access file should be housed on the server. The routing file should be housed on every Un*x machine. The DOS and, presumably, Macintosh computers will do their own routing.

8.2.1 The Access File

With socks4.2 Beta, the access file is called "sockd.conf". It should contain 2 lines, a permit and a deny line. Each line will have three entries:

- The Identifier (permit/deny)
- The IP address
- The address modifier

The identifier is either permit or deny. You should have both a permit and a deny line.

The IP address holds a four byte address in typical IP dot notation. I.E. 192.168.2.0.

The address modifier is also a typical IP address four byte number. It works like a netmask. Envision this number to be 32 bits (1s or 0s). If the bit is a 1, the corresponding bit of the address that it is checking must match the corresponding bit in the IP address field. For instance, if the line is:

```
permit 192.168.2.23 255.255.255.255
```

it will permit only the IP address that matches every bit in 192.168.2.23, eg, only 192.168.2.3. The line:

```
permit 192.168.2.0 255.255.255.0
```

will permit every number within group 192.168.2.0 through 192.168.2.255, the whole C Class domain. One should not have the line:

```
permit 192.168.2.0 0.0.0.0
```

as this will permit every address, regardless.

So, first permit every address you want to permit, and then deny the rest. To allow everyone in the domain 192.168.2.xxx, the lines:

```
permit 192.168.2.0 255.255.255.0
deny 0.0.0.0 0.0.0.0
```

will work nicely. Notice the first "0.0.0.0" in the deny line. With a modifier of 0.0.0.0, the IP address field does not matter. All 0's is the norm because it is easy to type.

More than one entry of each is allowed.

Specific users can also be granted or denied access. This is done via ident authentication. Not all systems support ident, including Trumpet Winsock, so I will not go into it here. The documentation with socks is quite adequate on this subject.

8.2.2 The Routing File

The routing file in SOCKS is poorly named "socks.conf". I say "poorly named" because it is so close to the name of the access file that it is easy to get the two confused.

The routing file is there to tell the SOCKS clients when to use socks and when not to. For instance, in our network, 192.168.2.3 will not need to use socks to talk with 192.168.2.1, firewall. It has a direct connection in via Ethernet. It defines 127.0.0.1, the loopback, automatically. Of course you do not need SOCKS to talk to yourself. There are three entries:

- deny
- direct
- sockd

Deny tells SOCKS when to reject a request. This entry has the same three fields as in sockd.conf, identifier, address and modifier. Generally, since this is also handled by sockd.conf, the access file, the modifier field is set to 0.0.0.0. If you want to preclude yourself from calling any place, you can do it here.

The direct entry tells which addresses to not use socks for. These are all the addresses that can be reached without the proxy server. Again we have the three fields, identifier, address and modifier. Our example would have

```
direct 192.168.2.0 255.255.255.0
```

Thus going direct for any on our protected network.

The sockd entry tells the computer which host has the socks server daemon on it. The syntax is:

```
sockd @=<serverlist> <IP address> <modifier>
```

Notice the @= entry. This allows you to set the IP addresses of a list of proxy servers. In our example, we only use one proxy server. But, you can have many to allow a greater load and for redundancy in case of failure.

The IP address and modifier fields work just like in the other examples. You specify which addresses go where through these. 6.2.3. DNS from behind a Firewall

Setting up Domain Name service from behind a firewall is a relatively simple task. You need merely to set up the DNS on the firewalling machine. Then, set each machine behind the firewall to use this DNS.

8.3 Working With a Proxy Server

8.3.1 Unix

To have your applications work with the proxy server, they need to be "sockified". You will need two different telnets, one for direct communication, one for communication via the proxy server. SOCKS comes with instructions on how to SOCKify a program, as well as a couple pre-SOCKified programs. If you use the SOCKified version to go somewhere direct, SOCKS will automatically switch over to the direct version for you. Because of this, we want to rename all the programs on our protected network and replace them with the SOCKified programs. "Finger" becomes "finger.orig", "telnet" becomes "telnet.orig", etc. You must tell SOCKS about each of these via the include/socks.h file.

Certain programs will handle routing and sockifying itself. Netscape is one of these. You can use a proxy server under Netscape by entering the server's address (192.168.2.1 in our case) in the SOCKs field under Proxies. Each application will need at least a little messing with, regardless of how it handles a proxy server.

8.3.2 MS Windows with Trumpet Winsock

Trumpet Winsock comes with built in proxy server capabilities. In the "setup" menu, enter the IP address of the server, and the addresses of all the computers reachable directly. Trumpet will then handle all outgoing packets.

8.3.3 Getting the Proxy Server to work with UDP Packets

The SOCKS package works only with TCP packets, not UDP. This makes it quite a bit less useful. Many useful programs, such as talk and Archie, use UDP. There is a package designed to be used as a proxy server for UDP packets called UDPrelay, by Tom Fitzgerald <fitz@wang.com>. Unfortunately, at the time of this writing, it is not compatible with Linux.

8.4 Drawbacks with Proxy Servers

The proxy server is, above all, a `security device`. Using it to increase internet access with limited IP addresses will have many drawbacks. A proxy server will allow greater access from inside the protected network to the outside, but will keep the inside completely unaccessible from the outside. This means no servers, talk or archie connections, or direct mailing to the inside computers. These drawbacks might seem slight, but think of it this way:

- You have left a report you are doing on your computer inside a firewall protected network. You are at home, and decide that you would like to go over it. You can not. You can not reach your computer because it is behind the firewall. You try to log into `firewall` first, but since everyone has proxy server access, no one has set up an account for you on it.

- Your daughter goes to college. You want to email her. You have some private things to talk about, and would rather have your mail sent directly to your machine. You trust your systems administrator completely, but still, this is private mail.
- The inability to use UDP packets represents a big drawback with the proxy servers. I imagine UDP capabilities will be coming shortly.

FTP causes another problem with a proxy server. When getting or doing an `ls`, the FTP server opens a socket on the client machine and sends the information through it. A proxy server will not allow this, so FTP doesn't particularly work.

And, proxy servers run slow. Because of the greater overhead, almost any other means of getting this access will be faster.

Basically, if you have the IP addresses, and you are not worried about security, do not use a firewall and/or proxy servers. If you do not have the IP addresses, but you are also not worried about security, you might also want to look into using an IP emulator, like Term, Slirp or TIA. Term is available from **ftp://sunsite.unc.edu**, Slirp is available from **ftp://blitzen.canberra.edu.au/pub/slirp**, and TIA is available from marketplace.com. These packages will run faster, allow better connections, and provide a greater level of access to the inside network from the internet. Proxy servers are good for those networks which have a lot of hosts that will want to connect to the internet on the fly, with one setup and little work after that.

9 Advanced Configurations

There is one configuration I would like to go over before wrapping this document up. The one I have just outlined will probably suffice for most people. However, I think the next outline will show a more advanced configuration that can clear up some questions. If you have questions beyond what I have just covered, or are just interested in the versatility of proxy servers and firewalls, read on.

9.1 A large network with emphasis on security

Say, for instance, you are the leader of millisha and you wish to network your site. You have 50 computers and a subnet of 32 (5 bits) IP numbers. You need various levels of access within your network because you tell your followers different things. Therefore, you'll need to protect certain parts of the network from the rest.

The levels are:

1. The external level. This is the level that gets shown to everybody. This is where you rant and rave to get new volunteers.
2. **Troop** This is the level of people who have gotten beyond the external level. Here is where you teach them about the evail goverment and how to make bombs.

3. **Mercenary** Here is where the *real* plans are keep. In this level is stored all the information on how the 3rd world goverment is going to take over the world, your plans involving Newt Gingrich, Oklahoma City, lown care products and what realy is stored in that hangers at area 51.

9.1.1 The Network Setup

The IP numbers are arranged as:

- 1 number is 192.168.2.255, which is the broadcast address and is not usable.
- 23 of the 32 IP addresses are allocated to 23 machines that will be accessible to the internet.
- 1 extra IP goes to a linux box on that network
- 1 extra goes to a different linux box on that network.
- 2 IP #'s go to the router
- 4 are left over, but given domain names paul, ringo, john, and george, just to confuse things a bit.
- The protected networks both have the addresses 192.168.2.xxx

Then, two separate networks are built, each in different rooms. They are routed via infrared Ethernet so that they are completely invisible to the outside room. Luckily, infrared ethernet works just like normal ethernet.

These networks are each connected to one of the linux boxes with an extra IP address.

There is a file server connecting the two protected networks. This is because the plans for taking over the world involves some of the higher Troops. The file server holds the address 192.168.2.17 for the Troop network and 192.168.2.23 for the Mercenary network. It has to have different IP addresses because it has to have different Ethernet cards. IP Forwarding on it is turned off.

IP Forwarding on both Linux boxes is also turned off. The router will not forward packets destined for 192.168.2.xxx unless explicitly told to do so, so the internet will not be able to get in. The reason for turning off IP Forwarding here is so that packets from the Troop's network will not be able to reach the Mercenary network, and vica versa.

The NFS server can also be set to offer different files to the different networks. This can come in handy, and a little trickery with symbolic links can make it so that the common files can be shared with all. Using this setup and another ethernet card can offer this one file server for all three networks.

9.1.2 The Proxy Setup

Now, since all three levels want to be able to monitor the network for their own devious purposes, all three need to have net access. The external network is connected directly into the internet, so we don't have to mess with proxy servers here. The Mercenary and Troop networks are behind firewalls, so it is necessary to set up proxy servers here.

Both networks will be setup very similarly. They both have the same IP addresses assigned to them. I will throw in a couple of parameters, just to make things more interesting though.

1. No one can use the file server for internet access. This exposes the file server to viruses and other nasty things, and it is rather important, so its off limits.
2. We will not allow troop access to the World Wide Web. They are in training, and this kind of information retrieval power might prove to be damaging.

So, the sockd.conf file on the Troop's linux box will have this line:

```
deny 192.168.2.17 255.255.255.255
```

and on the Mercenary machine:

```
deny 192.168.2.23 255.255.255.255
```

And, the Troop's linux box will have this line

```
deny 0.0.0.0 0.0.0.0 eq 80
```

This says to deny access to all machines trying to access the port equal (eq) to 80, the http port. This will still allow all other services, just deny Web access.

Then, both files will have:

```
permit 192.168.2.0 255.255.255.0
```

to allow all the computers on the 192.168.2.xxx network to use this proxy server except for those that have already been denied (ie. the file server and Web access from the Troop network).

The Troop's sockd.conf file will look like:

```
deny 192.168.2.17 255.255.255.255
deny 0.0.0.0 0.0.0.0 eq 80
permit 192.168.2.0 255.255.255.0
```

and the Mercenary file will look like:

```
deny 192.168.2.23 255.255.255.255
permit 192.168.2.0 255.255.255.0
```

This should configure everything correctly. Each network is isolated accordingly, with the proper amount of interaction. Everyone should be happy.

Now, take over the world!