Firewalling and Proxy Server HOWTO

David Rudder, drig@execpc.com

v0.2, 17 July 1995

This document is designed to teach the basics of setting up a firewall on a Linux based PC. Also covered is the installation and use of Proxy Servers to allow greater access to the Internet from behind a firewall.

Contents

1 Introduction **2**

1.1 Feedback 2
1.2 Disclaimer 2
1.3 Copyright 3
1.4 My Reasons for Writing This 3
1.5 TODO 3
1.6 Further Readings 3

2 Understanding Firewalls **4**

2.1 Drawbacks with Firewalls 4
2.2 Proxy Servers 5

3 Setting This All Up **5**

3.1 Hardware requirements 5
3.2 Setting up the Software 5
3.3 Configuring the Network Addresses 6
3.4 Testing it out 6
3.5 Securing the Firewall 6

4 Firewalling Software **7**

4.1 Available packages 7
4.2 The TIS Firewall Toolkit 7
4.3 TCP Wrapper 8
4.4 Ipfw and Ipfw Admin 8

5 The Proxy Server 8

5.1 Setting up the Proxy Server . . . 8

5.2 Configuring the Proxy Server . . . 8

5.2.1 The Access File . . . 8

5.2.2 The Routing File . . . 9

5.2.3 DNS from behind a Firewall . . . 10

5.3 Working With a Proxy Server . . . 10

5.3.1 Unix . . . 10

5.3.2 MS Windows with Trumpet Winsock . . . 11

5.4 Getting the Proxy Server to work with UDP Packets . . . 11

5.5 Drawbacks with Proxy Servers . . . 11

6 Advanced Configurations 12

6.1 A large network with emphasis on security . . . 12

6.1.1 The Network Setup . . . 12

6.1.2 The Proxy Setup . . . 13

1 Introduction

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. I am particularly looking for feedback from people who use Macintosh computers, as the information I have on them is scanty. **PLEASE PLEASE 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 drig@execpc.com.

1.2 Disclaimer

This document is meant as an introduction to how Firewalls and Proxy Servers work. I am not, not do I pretend to be, a security expert. I am just some guy who has read to much and likes computers more than people. I AM NOT RESPONSIBLE FOR ANY DAMAGES INCURRED DUE TO ACTIONS TAKEN BASED ON THIS DOCUMENT. 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

1.4 My Reasons for Writing This

There were a lot of discussions on comp.os.linux.* over the past year or so asking for Firewalling help. It seemed as if no one was going to answer these. I assumed that the reason was that no one knew how. So, I set aside some time to play with Firewalling and teach myself it. This document is in response to those requests.

1.5 TODO

- Learn how to do this on a Macintosh
- Learn different Windows TCP/IP packages
- Find a good UDP proxy server that works with Linux

1.6 Further Readings

- The Documentation for the TIS Firewall Toolkit
- The NET-2 HOWTO
- The PPP HOWTO
- The Ethernet HOWTO

- The Multiple Ethernet Mini HOWTO
- Networking with Linux
- TCP/IP Network Administrator's Guide by O'Reilly and Associates

The Firewall Toolkit by TIS has a collection of the best documentation I have found on firewalls and related constructs. For more on the `Firewall Toolkit`, look for the section *Firewalling Software*.

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 block from the passenger compartment. They are meant to protect the passenger in case the car explodes.

A **Firewall** in computers is a logical device that protects a private network from the public part. How they work is:

1. You take a computer that has routing capabilities (such as a linux box)
2. Put in 2 interfaces (ie Serial ports, Ethernet, Token Ring, etc)
3. Turn *off* IP forwarding
4. Connect the internet to one interface
5. Connect the protected network to the other interface

Now, you have two distinct networks, which share a computer. The firewall computer, from now on named `"firewall"`, can reach both the protected network and the internet. The protected network can not reach the internet, and the internet can not reach the protected network.

For someone to reach the internet from inside the protected network, one must telnet to `firewall`, and use the internet from there. Accordingly, in order to get into the protected network, one must go through the firewall first.

This provides excellent security against attacks from the internet. If someone wants to make a concerted attack against the protected network, they must go through the firewall first, making it a two-step, and thus much harder, attack. If someone wants to attack the protected network via a more common method, such as mail bombing, or the infamous `"Internet Worm"`, they will not be able to reach the protected network. This makes for excellent protection.

2.1 Drawbacks with Firewalls

The biggest problem with firewalls is that they greatly inhibit the access to the internet from the inside. Basically, they reduce the usage of the internet to that which one would have via a dial-up shell account. Having to login to `firewall` and then do all internet access is a severe restriction. Programs like *Netscape*, which require a direct internet connection, will not work from behind a firewall. The answer to these problems is having a **Proxy Server**.

2.2 Proxy Servers

Proxy servers are constructs that allow direct internet access from behind a firewall. How they work is they open a socket on the server, and allow communication via that socket to the internet. For example, if my computer, `drig` is inside the protected network, and I want to browse the Web using *Netscape*, I would set up a proxy server on `firewall`. The proxy server would be configured to allow requests from my computer, trying for port 80, to connect to its port 1080, and it would then redirect all requests to the proper places.

Anyone that has used *TIA* or *TERM* has seen this concept before. Using these two programs, you can redirect a port. A friend had TIA setup to allow anyone using 192.251.139.21 port 4024 to connect to his Web Server. The proxy server works like this, but backwards. To connect to anyone else's port 80, you must use port 1080 (or whichever port you set it for).

The great thing about proxy servers is that they are completely secure, when configured correctly. They *will not* allow someone in through them.

3 Setting This All Up

3.1 Hardware requirements

For our example, the computer is a 486-DX66, 8 megs of memory, 500 megs Linux partition, with a PPP connection to it internet provider over a 14.4 modem. This setup is your basic Linux box. To make it a firewall, we add one NE2000 Ethernet card. It is then connected to 3 PC's running Windows 3.1 with Trumpet Winsock and 2 Suns running SunOS 4.1. This setup was chosen because it is a fairly common and they are both platforms that I am familiar with. I imagine much of the same stuff that I am talking about here is doable with Macs, but since I don't use Macs frequently enough, I don't really know.

3.2 Setting up the Software

So, you have one Linux box connected to the net via a 14.4 PPP line. You then have a Ethernet network connected to the Linux box and all the other computers. First, you must recompile the Linux kernel with the appropriate options. At this point, I would look at the **Kernel HOWTO**, the **Ethernet HOWTO**, and the **NET-2 HOWTO**. Then, do a `"make config"`:

1. Turn on Networking Support
2. Turn on TCP/IP Networking
3. Turn off IP Forwarding (CONFIG_IP_FORWARD).
4. Turn on IP Firewalling
5. Probably turn on IP accounting. Seems prudent since we are setting up a security measure
6. Turn on Networking Device Support
7. We turn on PPP and Ethernet support, but that depends on your interfaces

Then, we recompile, reinstall the kernel and reboot. The interfaces should show up in the boot-up sequence, and we should be fine. If not, go over the other HOWTOs again until it is working.

3.3 Configuring the Network Addresses

This is the real interesting part. Since we do not want the internet to have access, we do not need to use real addresses. One good C Class to use is 192.168.2.xxx, which was set aside as a dummy test domain. Thus, no one uses it, and it will not conflict with any requests for the outside. So, in this configuration, only one real IP address is needed. The others are free for the taking and will not affect the network at all.

Assign the real IP to the serial port used for the PPP. Assign 192.168.2.1 to the Ethernet card on `firewall`. Assign all the other machines in the protected network some number in that domain.

3.4 Testing it out

First, try to ping the internet from `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.

Now, try pinging between hosts within the protected network. 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, every machine in the protected network should be able to ping `firewall`. If not, go back again. Remember, they should be able to ping 192.168.2.1, not the PPP address.

Then, try to ping the PPP address of `firewall` from inside the protected network. If you can, then you have not turned off *IP Forwarding* and you will have to recompile the kernel. Having assigned the protected network the 192.168.2.1 domain means that no packets will be routed to this network anyway, but it is safer to have IP Forwarding turned off anyway. This leaves the control in your hands, not in the hands of your PPP provider.

Finally, ping each machine within the protected network from `firewall`. By this time, there should be no problems.

Now, you have your basic firewall setup.

3.5 Securing the Firewall

The firewall is no good if it is left wide open to attacks. First, look at /etc/inetd.conf. This file is what is called a "super server". It runs a bunch of the server daemons as they are requested. Examples are:

- Telnet
- Talk
- FTP

- Daytime

Turn off everything that is not needed. Definitely turn off netstat, systat, tftp, bootp, and finger. You might also want to turn off telnet, and only allow rlogin, or vica-versa. To turn a service off, merely put a # in front of it. Then, 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.

4 Firewalling Software

4.1 Available packages

Strict firewalling does not need any software except the Linux kernel and the base networking packages (inetd, telnetd and telnet, ftpd and ftp). But, a firewall like this is extremely restrictive and not very useful.

So, software packages have been made to make a firewall more useful. The one that I would like to examine in most detail is a package called "*socks*" which is a `proxy server`. But, there are two other programs that you might want to keep in mind. I would like to give you a short review of them now.

4.2 The TIS Firewall Toolkit

TIS 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 better contrast the two, lets take 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 difference, 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**.

4.3 TCP Wrapper

TCP wrapper is not a firewalling utility, but it allows for many of the same effects. Using TCP wrapper, you can control who has access to your machine and to what services as well as keep logs of the connections. It does basic forgery detection also.

TCP wrapper is not covered more extensively here because of a couple of reasons.

- It is not a real firewall.
- To use it, you have to be connected to the Internet, thus you have to have an IP address.
- It only controls the machine it is installed on, and is thus not good for a network. Firewalls can protect every machine of every architecture. TCP wrapper won't work on Macintoshes and MS Windows machines.

4.4 Ipfw and Ipfw Admin

5 The Proxy Server

5.1 Setting up the Proxy Server

The proxy server requires additional software. You can get this software 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 in making it. Make sure that the Makefiles are correct. Some are, some are not.

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.

5.2 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.

5.2.1 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. Ie 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`

then, 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.

5.2.2 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.

5.2.3 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.

5.3 Working With a Proxy Server

5.3.1 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.

5.3.2 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.

5.4 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.

5.5 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 `get`ing 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.

6 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.

6.1 A large network with emphasis on security

Say, for instance, you are the leader of the *Milwaukee 23rd Discordian Cabal*. You wish to network your site. You have 50 computers and a subnet of 32 (5 bits) IP numbers. You have various levels of access. You tell your disciples different things according to each level. Obviously, you would want to protect certain parts of the network from the disciples that are not in that level.

Disclaimer: I am not a member of the Discordians. I do not know their terminology, nor do I really care. I am using them as an example only. Please send all flames to

The levels are:

1. **The external level**. This is the level that gets shown to everybody. Basically, this is the ranting and raving about Eris, Goddess of Discord, and all the rest of the drivel.
2. **Sage** This is the level of people who have gotten beyond the external level. Here is where you tell them that discord and structure are really one, and that Eris is also Jehovah.
3. **Adept** Here is where the *real* plan is. In this level is stored all the information on how the Discordian Society is going to take over the world through a devious, yet humorous, plan involving Newt Gingrich, Wheaties Cereal, O.J. Simpson, and five hundred crystals, all erroneously marked "6.5 MHz".

6.1.1 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 (or so i think), so we can just think of them like normal.

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 sages. The file server holds the address `192.168.2.17` for the sage network and `192.168.2.23` for the adept 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 sage network will not be able to reach the adept 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.

6.1.2 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 adept and sage 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 Sage 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 sage linux box will have this line:

```
deny 192.168.2.17 255.255.255.255
```

and on the adept machine:

```
deny 192.168.2.23 255.255.255.255
```

And, the sage 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 sage network).

The sage 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 adept 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, *look out for your 6.5 MHz crystals*