▶▶Debian Firewalls ◀◀

This is a step by step guide for setting up a custom Debian firewall for your home or office network.

~ ~ ~

Updated 1/11/2006

How did this tutorial work for you? Please leave FEEDBACK!

Did this tutorial help you? Please consider a donation to help support expanding it!

A tutorial by:
Matt LaPlante
CCNP, CCDP, A+, Linux+, Cisco Firewall Specialist
http://www.cyberdogtech.com/firewalls/

Debian Firewalls - Introduction

Most networks these days run behind some type of gateway/router/firewall device. Home office and SOHO networks often use small firewall/routers made by companies such as Linksys, D-Link, and Netgear to provide their network connectivity. The problem is these devices are often weak, underpowered, and feature limited. The solution? Building your own!

Linux is a wonderfully powerful and versatile operating system. It can run anything from a full-fledged office workstation, to a server cluster, to a cell phone. You can also conquer the power of Linux to create a powerful firewall and router for your home network. Best of all, the hardware required is cheap, and the software is free.

Why Debian? Ultimately it comes down to personal preference. Most any Linux distribution will work. I've built firewall systems with several Linux distros now, and I've chosen Debian as my favorite for a few reasons:

- Minimal builds are fairly minimal. The "minimal" build on some distros is a lot larger than on others. It all has to do with how many packages are tied to the core OS. In this case, smaller is better.
- Huge package selection. Debian has one of the largest selections of pre-built packages available with the OS. Provides lots of versatility with minimal effort.
- Supports customization well. While all distros are customizable, some are inherently more or less flexible than others. Debian generally shows very little resistance when you decide to go off the beaten path.
- Excellent support. Debian has a huge community. Security updates are released very quickly, and there's always somebody out there to ask for help.

Alternative Alert! - Distributions
This guide uses the Debian Linux distribution. One of the greatest (and potentially most complicated) parts of Linux is the variety of operating systems available. The software used in this guide is all free and open source, so with a little adaptation, you can create a Linux firewall with any brand you desire.

Things to know before we start: The purpose of this guide is not to be the end all be all of Linux firewalls, it's merely a starting point. You're encouraged to go above and beyond the bare minimums shown here. The possibilities are truly endless, but to get the most out of your machine you must not be afraid to explore!

Proceed to Step 1 - Installation

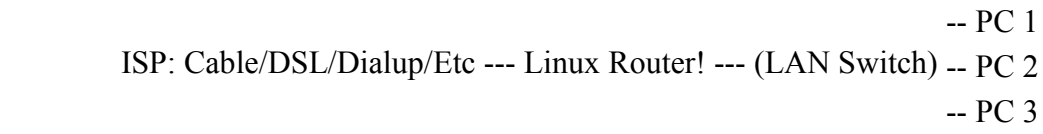
Debian Firewalls -- Installation

System Requirements: A Linux firewall is a great way to recycle old computer hardware! You can get a suitable system off of eBay for under $50, or perhaps even free from your company's dumpster. :) I recommend the following specs:

- Memory: 64-128 mb
- Hard disk: 2-3 gb
- Processor 200mhz+
- 2+ Network Interfaces
- CD-ROM or Floppy drive
- Some type of video output
- A network switch.

Keep in mind that these are very rough specs. You may want more or less depending on what exactly you plan on doing with your machine. A bare minimum firewall will probably run fine on even lower specs. On the other hand, if you want to do some heavy lifting like IPSec or proxying, you may want a heftier machine. Of course, you can always upgrade later.

You will need a simple network switch to allow you to connect all the computers on your LAN to your new router. Our setup assumes the following topology:



Now head over to www.debian.org. Under "Getting Debian" click CD ISO images. You can choose to download any of the CD (or Floppy) types. A minimal image is all that's necessary, although the full discs will work just as well. Use your favorite CD burning application to create the installation CDs. There's lots of documentation on the site if you need help with this.

Boot your system from the first installation CD. You should be presented with the Debian installation screen. By default Debian installs using the 2.4 kernel series. The kernel is the core of the Linux OS. I personally recommend using the 2.6 series, which is the latest. The 2.4 series, while being more mature and potentially more stable, is now considered legacy.

Type "linux26" at the *boot:* prompt and hit enter.

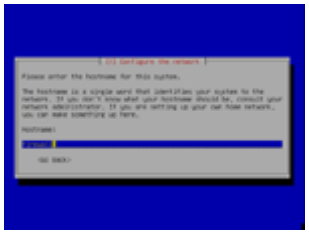
Now choose your language of choice. In my case, English.

...country and region...

...and keymap...

The installer will now try to detect your hardware, and hopefully be successful. Assuming you have multiple network cards, and are using the network installation CDs or Floppys, the installer will ask you to select which card is connected to your internet connection. If DHCP fails, make sure you've connected the right interface to your ISP or current router and try again.

Now, enter a hostname (computer name) for your system. You can call it whatever you want.



If your home network isn't part of a domain, or you're not sure what this is, leave the next screen as it is and hit Enter. You'll know if you need to

change it.

Now we need to partition the hard disk. **This will erase everything on the hard drive! Don't do it unless you're ready to lose anything on the drive!** Hit Enter to "Erase Entire Disk" and wipe the disk clean.

Since this drive won't need to contain much in the way of programs, we can safely choose the first option, "All Files in One Partition."

The partitioner will automatically create your root and swap partitions. The defaults should be fine, but you may also choose to use different file systems or partition sizes if you know what you're doing. Once you're happy with it, hit Finish.

Select Yes to Write the changes to disks. This is your **last chance** to not erase your drive!

The partitioner will finish setting up the partitions and start installing the system base. This will take a couple minutes. Hit Yes to install the Grub bootloader.

Setup is almost complete! Remove your CDs and Floppys and hit Continue to boot into your new system.

Proceed to Step 2 - Initial Configuration

Debian Firewalls -- Initial Configuration

Congratulations! If you've made it this far, you should have a working Debian installation on your new router. Now comes the fun part...setting up the network services. Go ahead and read the welcome screen, then hit OK. Next, answer the question regarding your system clock...

...select your time zone...

Now enter a root password. As you should know, root password is the Unix/Linux equivalent to the system administrator. This password will allow full, unrestricted system usage. For this reason you should try to make the password as strong as possible.

Now we'll create a standard user account. This is a "non-privileged" user which only has limited ability to change the system. This is the account you should use normally to avoid accidentally breaking the system. Enter the actual name, the user name, and the password as it prompts you.

Next we move on to configuring Apt. Apt is the Debian package manager...if you plan on using Debian you should learn a bit about Apt. I recommend the documentation at www.debian.org for that. For now, you can scan your CDs if you have them, otherwise click No when you're done.

Select Yes to add another Apt source.

Apt packages can be downloaded directly from the Debian servers, and since I chose to use a network install, I have to add the network Apt sources. You can choose either HTTP or FTP to do this:

Choose the country nearest you, then choose any mirror site that you like. They all function the same.

Unless your ISP uses a proxy, you can just leave the next field blank and hit Ok. If you don't know what this is, you probably don't need it.

Apt will now download the list of packages from the server you've chosen. We will now be prompted to choose which pre-configured system installation we want. We want to choose ONLY *manual package selection*! This is the key to building a custom system, as it will allow is to have a bare system and install only the programs we need.

We are now presented with the Aptitude program interface. Aptitude is a very handy package manager that allows you to browse all the Debian packages available for install. If you know in advance that you want certain packages, you may do so here. For the sake of this tutorial, we are simply going to quit the program for now. Type the lowercase letter 'q' to quit, and select Yes to close the program.

On the next screen just select Ok to local delivery only unless you know you want to use different SMTP settings.

You should be able to just hit Ok for the next screen as well.

Congratulations! Initial configuration is now complete and you have a fully working system. Now we can get on to the fun stuff...setting up our network services.

Proceed to Step 3 - Apt, Packages, and Updating

Debian Firewalls -- Apt, Packages, & Upgrading

You should now have a login screen for your new Debian router. Go ahead and login with the user account you created. Before we continue, you should familiarize yourself with the Debian package management system, also known as Apt.

Debian "packages" are simply pre-compiled software programs designed to work with your Debian system. The easiest (although not only) way to install these packages is using the apt program, which comes preinstalled on the system. For the sake of simplicity, we'll be using Debian packages for the remainder of this tutorial.

Alternative Alert! - Packages
The wonderful thing about open source software is not only is it free, but the source code is available for download by anyone who wants it. All of the programs featured here can be downloaded individually as source code rather than as Debian packages. Compiling programs by hand has several advantages, including greatly increased customizability, as well as improved performance. Compiling source also has several downsides: First, you need to know what you're doing to compile it in the first place. You also need to be willing to re-compile the source every time a security or bug fix update is released. Packages, on the other hand, can be installed and updated with a couple commands, and are maintained by dedicated members of the Debian community. For simplicity, we'll be using packages for this tutorial, but it in no way implies you can't or shouldn't compile your own program if you know how!

First, we need to know how to update the local Apt cache. The Apt cache stores a local database of all available packages on your system. You should update it every time you work with Apt. To update it, run *apt-get update* from the command line:

Firewall:~# apt-get update

Now that your local cache is updated, the first thing you should do is update all the currently installed applications on your system. This is accomplished by running *apt-get upgrade*:

Firewall:~# apt-get upgrade

Apt will check if any of your installed programs are outdated. Say Y (yes) if it offers you any upgrades...never turn down upgrades!!

You should run *apt-get update* and *apt-get upgrade* REGULARLY. This keeps all installed packages on your system up to date. Failure to do so may leave your system vulnerable to security flaws. Your entire network could be compromised if your software is not up to date!

You'll also want to know how to locate new packages that you can install on your system. Exploring is good! To search for a package, use the *apt-cache search* command:

Firewall:~# apt-cache search [searchterm]

You can use the *apt-cache show* command to get details about a package:

Firewall:~# apt-cache show [appname]

To install a package, use *apt-get install*:

Firewall:~# apt-get install [appname]

To remove a package, use *apt-get remove*:

Firewall:~# apt-get remove [appname]

Finally, the *aptitude* program will give you a next text-based interface for searching, installing, and removing packages. If you're a menu-driven person, try it by running *aptitude* from the command line:

```
Firewall:~# aptitude
```

Proceed to Step 4 - Interfaces

Debian Firewalls -- Interfaces

First, we have to configure our network interfaces. For the sake of this tutorial, we'll be using two ethernet interfaces. One is the internal interface, and connects to our LAN switch. The second interface is external and connects to our ISP (cable modem/DSL modem/etc).

Your LAN can use any private IP address range. We'll be setting up PAT (Port Address Translation, also known as NAT - Network Address Translation) shortly. This will allow all the computers on our local network to use internal addresses. The firewall will automatically translate our internal address into the external address assigned by our ISP.

Interfaces are configured in the */etc/network/interfaces* configuration file. We'll be using the *nano* text editor throughout this tutorial. If you prefer another editor such as vi or emacs, feel free to use that instead. Open the networking configuration file in nano:

Firewall:~# nano -w /etc/network/interfaces

Hopefully the interfaces file will already have your ISP interface preconfigured from setup. You'll also have a loopback interface (lo) which is normal. We'll assume for the rest of this tutorial that eth0 is the name of your ISP interface and eth1 is the name of your internal (LAN) interface. If you're not sure what your interfaces are named, run the *ifconfig* command from the command line...they should be listed.

The actual configuration file is quite simple. Lets break it down:

auto eth0 - automatically start the interface when the system boots. You'll want this for any vital interfaces.

iface eth0 inet dhcp - declares that the eth0 interface should get its information from DHCP. This is what you'll want if you automatically get your IP Address from your ISP.

Now we have to add or edit our internal interface, which will have the static IP address of 192.168.1.1:

```
auto eth1
iface eth1 inet static
address 192.168.1.1
netmask 255.255.255.0
```

These entries tell the system to automatically start eth1 when the system boots, and use a static ip address of 192.168.1.1 for the interface.

Once you've made the necessary changes, hit Ctrl+x to close nano, and save the file.

Finally, it's good to know how to change interfaces between up and down state. To bring an interface online manually, use *ifup [interface name]*:

Firewall:~# ifup eth0

To shut an interface down, use *ifdown [interface name]*:

Firewall:~# ifdown eth0

Alternative Alert! - Interfaces
Linux supports a very broad variety of network interfaces. It can run practically anything including ethernet, dialup, wireless, token ring, atm, and more. This will leave you the ability to get very creative with your network topology if you so choose. You can add any kind of connection you like, including in combinations that provide redundancy and multi-homing. For the sake of simplicity, this tutorial only covers a basic ethernet configuration.

Proceed to Step 5 - The Firewall

Debian Firewalls -- The Firewall

Version 2.6 of the Linux kernel uses *iptables* to provide its firewall facilities. For more information on iptables, see http://www.netfilter.org. Iptables is a wonderfully robust and functional firewall package, and will form the core of all of our machine's firewall and routing functionality. Iptables is installed by default as part of the minimal Debian installation, so there's no further installation needed.

The predecessor to iptables, *ipchains* is also installed by default, but we don't need it on our system. I recommend removing it to keep things simple and uncluttered:

Firewall:~# apt-get remove ipchains

Iptables is wonderfully powerful, but unfortunately that power comes at a price...namely configuration. While it can technically be configured by hand, it's a common saying that iptables configuration is not human readable. In short, it's very complex and can quickly become overwhelming. Luckily, we have a solution in the form of a program called *Shorewall*. Shorewall was written by Tom Eastep, and is available via http://shorewall.sourceforge.net. Of course, we also have a Debian package for shorewall, so there's no need to download and install it by hand. To start, use apt-get to install the shorewall package:

Firewall:~# apt-get install shorewall

At this point apt may tell you it has to install a couple extra supporting package along with shorewall. This is normal and you should accept the prompt to allow it to install everything. Don't worry about the "suggested" packages, we won't need those.

Alternative Alert! - Shorewall
The problem of configuring iptables is not new, and as with most things in computing, there's more than one way to solve the problem. While this tutorial uses Shorewall to do its dirty work, there are other programs out there that will do the same. One popular alternative is *FireHOL* (http://firehol.sourceforge.net). If shorewall isn't floating your boat look around...there are always alternatives.

Before we move on, let's clear up a couple common misconceptions: Shorewall is not a firewall, and in fact it's not even an application. The common notion of a program (or daemon) is that of an application that runs continuously. This is not the case with Shorewall. Instead, Shorewall is actually just a very large set of scripts which run once and then exit. Shorewall itself does not perform any firewalling work; it merely configures iptables to your specifications, then quits.

Now on to configuration. You probably noticed a warning message at the end of the Shorewall installation telling you the program will not start unless you change the /etc/default/shorewall file. Lets do that now:

Firewall:~# nano -w /etc/default/shorewall

Now simply change
startup = 0
to
startup = 1
save, and exit.

Shorewall configuration files are stored in two separate places:
/etc/shorewall stores all the program configuration files.
/usr/share/shorewall stores supporting files and action files.

On the Debian package version of shorewall, /etc/shorewall is rather empty. Luckily, we're provided with default configuration files in
/usr/share/doc/shorewall/default-config

Since we will need to use these config files to actually make Shorewall work, the first thing to do is to copy them over to /etc/shorewall:

Firewall:~# cp /usr/share/doc/shorewall/default-config/* /etc/shorewall/

Now our /etc/shorewall directory should have default copies of all the config files. Next we modify a few of them to get our firewall in basic working order. I'm only going to cover the basic configurations necessary to get the firewall working. **Please** read the documentation in each config file you edit so you can fully understand what each step is really doing!

First we add our network "zones" Shorewall uses zones as a way of defining different portions of our routed network. Our simple setup will have two zones: local and internet (loc & net). Shorewall can easily be extended to support many more zones such as a DMZ or a VPN zone. This configuration is performed in /etc/shorewall/zones:

Firewall:~# nano -w /etc/shorewall/zones

All we have to do here is name our zones:

net Net The Internet
loc Local Local Network

That's it, save and exit.

Next, we have to add our physical interfaces. This is done via /etc/shorewall/interfaces:

Firewall:~# nano -w /etc/shorewall/interfaces

First we add our internet (net) interface, eth0. We set it to automatically detect the interface settings, and also tell it to perform some filtering. I won't go into detail here; everything is explained nicely at the top of the file.

net eth0 detect dhcp,routefilter,tcpflags,nobogons

Second is our internal interface (eth1):

loc eth1 detect dhcp

Notice it's not generally necessary to turn on all the filtering options on the internal interface, as our local zone is considered "safe". We're only super cautious about traffic coming into the system from outside. Interface configuration is done, so save and close the file.

Our system uses PAT (port address translation). This is featured as the default on most small home and SOHO firewall devices. Basically, PAT allows our router to translate between our external IP address (on eth0) and all our internal addresses (connecting to eth1). This feature is often referred to (incorrectly) as NAT, or Network Address Translation. Please note that PAT/NAT are not required to operate a firewall, but you will have to set up alternative methods of routing instead. In Shorewall, PAT is configured in /etc/shorewall/masq:

Firewall:~# nano -w /etc/shorewall/masq

We have to tell shorewall that we want all traffic coming from inside the network (on eth1) to be translated out through the interface on eth0). We do this simply by specifying the interfaces:

eth0   eth1

It is important to note that as always, there are more advanced possibilities here than what we're using...read the documentation! Also, don't be fooled by /etc/shorewall/nat. This file is for providing Network Address Translation, which translates internal IP addresses to external IP addresses directly, rather than using a single external address and translating the ports. I recommend Wikipedia and Google if you want to learn more.

Now comes the ever important firewall policy. The policy forms the basis for how all traffic on our network will be treated. This is not for fine grained control, we'll get to that later. This just sets the baseline actions for a zone.

Firewall:~# nano -w /etc/shorewall/policy

We trust our local traffic, so we want to accept traffic that comes from our LAN, no matter where it's going:

loc all ACCEPT

Note that there is also a built-in zone for the firewall machine itself, which is "fw" by default. You have the option of adding that to the policy as well. While this rule is optional, I recommend adding it. It says all traffic generated by the machine is allowed. You can also leave this out and configure a more strict rule (in the rules file) to only allow certain traffic, but keep in mind if you do not add some type of policy or rule, you will not be able to use any network based features of the system. For example apt will no longer

work if fw traffic is not ACCEPTed. The rest of this tutorial assumes this option is added.

fw all ACCEPT

We **don't** trust external traffic from the internet. When we see internet traffic that doesn't match any specific rules (later), we want it DROPPED:

net all DROP

Finally, any traffic not matching the above is rejected (this must be the last rule):

all all REJECT

Finally we get to the last necessary file, /etc/shorewall/shorewall.conf. This file manages global shorewall options, and you should read it through completely.

Firewall:~# nano -w /etc/shorewall/shorewall.conf

Most importantly, change IP_FORWARDING to "On". If you don't, your packets won't be able to get out of the local network:

IP_FORWARDING=On

For security, I suggest setting DROPINVALID to "Yes" as well:

DROPINVALID=Yes

Read through the whole file and customize it as you wish. When you're done, save your work. That should complete the basic firewall configuration. You should run "shorewall check" to see if you've made any typos. It won't catch all possible errors, but it helps:

Firewall:~# shorewall check

If you get "Configuration Validated" you can go ahead and start Shorewall:

Firewall:~# /etc/init.d/shorewall start

Note that Shorewall should run automatically every time the system boots, so you won't have to do it manually. If you want to change your settings without rebooting, just use "restart" instead of start in the above command.

Proceed to Step 6 - Firewall Rules

Debian Firewalls -- Firewall Rules

The firewall we've set up so far is quite strict, it won't allow any traffic in at all. Often, there are times when we need to make exceptions to the firewall policy we set up early. To accomplish this, we use the /etc/shorewall/rules file.

Firewall:~# nano -w /etc/shorewall/rules

Say we have an internal web server that we want the world to be able to access. Well right off the bat we have two problems: First, the firewall policy denies any traffic initiated by the internet. Second, our internal web server has an internal address that's not associated with our global (outside) IP address. We can overcome both these problems by making a DNAT (dynamic NAT) rule in shorewall. For this example, lets assume our web server's internal address is 192.168.1.100.

DNAT net loc:192.168.1.100 tcp 80

Let's break this down: We're telling the firewall to use DNAT, which will dynamically redirect a request to our external address to one of our internal machines. *net* says this applies to traffic coming from our "net" zone. *loc:192.168.1.100* tells the firewall to send this traffic to a specific machine in our local zone, 192.168.1.100. Then we have to specify the protocol (tcp,udp,etc), and the destination port number (80 for http, etc). You can set any protocol and port number supported by the kernel. Now, when the firewall receives traffic to its external interface with a destination port of 80, it will automatically translate the traffic and send it to our internal web server. Pretty cool.

Next, let's say we have a server running on the firewall machine itself, for example SSH. Rather than deny or redirect connections, we want the firewall to pass the traffic to one of it's own servers. For this we use the ACCEPT option:

ACCEPT net fw tcp 22

This example will tell the firewall to accept any connections coming from the internet (net) zone to itself on port 22 (SSH). In other words, you can connect to an SSH server on your firewall with this rule.

Warning: Be careful when authorizing any inbound traffic! Rules such as this can be useful, but they can also be **very dangerous**. Any time you leave open services on the internet, you're making yourself vulnerable to attack. If a malicious person can connect to your ssh server at all, then there's a chance they can also get full access to your machine. Only use such options if you're well aware of the potential consequences!

Another popular option would be to allow people to ping your firewall, which is disabled by our default policy. To do this, allow the firewall to accept ICMP protocol traffic from the net zone:

ACCEPT net fw icmp

There are many potential applications for rules, even in basic usage. If you're a P2P user, look up what ports your favorite P2P app uses for incoming traffic, and create a DNAT rule to your local machine. This will allow you to get full connectivity to the P2P network without the hassles of the software knowing you're behind a firewall.

By default, whenever the firewall drops unauthorized traffic it is logged. To view the logs, run *less /var/log/syslog*. If you have traffic that is

constantly bombarding your firewall and filling up your logs, you can use the rules to silently drop the traffic without logging it. Just create a DROP entry:

DROP net fw udp 1026:1029

This will cause the firewall to drop any incoming traffic to the firewall on udp ports 1026-1029 without logging it. You can set any protocol and port numbers you wish.

Once you've finished setting up all your rules, don't forget to save the file, run *shorewall check*, and restart shorewall to apply the new configuration.

Finally, as always, read all of the options in the rules file. It's a very powerful tool, and you should study how to configure it properly.

Proceed to Step 7 - DNS

Debian Firewalls -- DNS

Now that we have a functioning firewall, lets add some basic network services to get our network really running at full throttle. First off, we can add a DNS proxy. This step is totally optional.

Basically a *proxy* works as a "man in the middle" between the client and the server computers. Proxies serve many purposes, some of the most common of which are providing fast access to popular material, and allowing filtering of data. The purpose of the DNS proxy is rather straight forward. Rather than having all the computers on the local network send traffic to our ISP's DNS servers, we have the local machines send their requests to our firewall. Our firewall then looks up the DNS information on the behalf of the machines, and returns the information to them. This allows for one specific benefit: speed. Our server is often referred to as a *caching* name server, because it stores the DNS records after it looks them up. Even if clients on your network request www.debian.org 20 times, the firewall only has to request the information from the ISP once...it uses its own memory to answer the other 19 queries. And since the local network is always faster than the internet connection, this should boost the speed of our DNS lookups. For our DNS Proxy we'll be using the *dnsmasq* package (http://www.thekelleys.org.uk/dnsmasq).

Alternative Alert! - DNSMasq
Once again, DNSMasq is just one of the potential solutions to the problem; there are several other options available. One popular one might be BIND (the Berkeley Internet Name Domain). In fact, just about any DNS server will fill the role. DNSMasq was chosen because it's lightweight, easy to configure, and was create with just this job in mind.

Let's get started by installing dnsmasq:

```
Firewall:~# apt-get install dnsmasq
```

With dnsmasq we only have one configuration file: /etc/dnsmasq.conf

```
Firewall:~# nano -w /etc/dnsmasq.conf
```

As is the running theme, there's lots of options here we won't touch on. Read the documentation completely so you know what the package is capable of. DNSMasq should work for us straight "out of the box" so there's no need to change any configurations unless you want to. If you change the config, restart it up with:

```
Firewall:~# /etc/init.d/dnsmasq restart
```

Debian Firewalls -- DHCP

DHCP is another one of our key services. The Dynamic Host Control Protocol as it is known, is the service that assigns all the local IP addresses to our internal computers. This is the default for most firewalled networks. To start off, we'll install DHCPD, the DHCP daemon:

Firewall:~# apt-get install dhcp

Alternative Alert! - DHCPD
You guessed it, more options. We've got loads of alternatives here, just do an *apt-cache search dhcp* if you don't believe me. Other available options are dhcp3-server, the newer version of the ISC we're going to use, as well as others. Even DNSMasq that we installed earlier can do DHCP serving if you wish.

Our configuration file for this daemon is /etc/dhcpd.conf

Firewall:~# nano -w /etc/dhcpd.conf

If you're running a network on a domain you can set the *domain-name* option. If not, or you don't know what it is, comment it out. As usual, any line starting with a # is a comment, and won't be processed.

The DHCP server configuration file has two "sections". The global options are configured outside of everything else, and are applied by default to any subnets you may configure. Then there are the "subnet" subsections. The rules in these sections only apply to the specific subnet in question, and override the global settings.

For starters, let's make the default-lease-time and max-lease-time longer. These settings simply tell the server and the client how long an address is valid on the network. After this time, the address must be renewed with the server. On a corporate network with lots of changing hosts, this value should be smaller to allow for better use of the address space. On our home network, computers rarely change, so a small number just creates extra traffic. Make the number larger reduces network overhead since computers can keep their addresses longer. I use the following:

```
default-lease-time 86400;
max-lease-time 604800;
```

This should default to one day and one week respectively.

Now we have to configure our local subnet. If you remember, our internal IP address was 192.168.1.1, meaning all our local computers should be on the 192.168.1.0/24 subnet. Here's the full configuration:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.10 192.168.1.99;
   option domain-name-servers 192.168.1.1;
   option netbios-name-servers 192.168.1.110;
   option routers 192.168.1.1;
}
```

Let's break this down: the first line declares that we're servicing subnet 192.168.1.0/24, our local network. These rules override global rules and apply only to that network.

The 1st option, *range* tells the server what range of addresses it should assign from. Here, clients will receive addresses between 192.168.1.10 and 192.168.1.99 inclusively.

The 2nd option, *domain-name-servers*, tells the hosts what DNS servers to use. This option gives hosts the firewall's address. This is assuming you have a DNS proxy set up as we did in the last step with DNSMasq. If there is no DNS service running on your firewall, you will need to use your ISPs DNS server addresses instead. You can also skip this step entirely and manually configure every client with DNS settings.

The 3rd rule, *netbios-name-servers* specifies what are often better known as WINS servers. These servers allow local clients to look up local machines based on their netbios names. This option should be left out unless you have a WINS/Netbios server on your local network. If you don't know what this is, skip it.

The final option, *routers*, is very important. It specifies the address of your gateway, 192.168.1.1. This tells all hosts which machine is the default gateway for the network, ie where to send all outgoing traffic.

Don't forget to close the subnet with a closing bracket "}".

By nature, DHCP may assign any address in the pool to any machine. Say you have a machine whose address you want to remain constant over time. Our DHCP server can accommodate that as well. To do so, we have to create a *host* record:

```
host webserver {
   hardware ethernet 0:0:A0:B1:D3:C4;
   fixed-address 192.168.1.130;
}
```

This creates a static host entry. It should go *outside* of any subnets in the configuration file. The *hardware ethernet* line **must** match the MAC address of the computer you want the IP address assigned to. The *fixed-address* portion simply assigns the given address to the computer with the matching MAC.

Finally, you'll want to tell DHCP which interface to listen to on startup. This can be done in /etc/default/dhcp.

Firewall:~# nano -w /etc/default/dhcp

Look for the INTERFACES= line near the top. Make sure the interface between the quotes matches the name of our internal interface, not our internet interface.

INTERFACES="eth1"

Once that's taken care of, you can start DHCPD.

Firewall:~# /etc/init.d/dhcp start

Your hosts should now be able to pull valid local addresses from the firewall. If you get any errors, check /var/log/syslog to see what went wrong.

Proceed to Step 9 - SSH

Debian Firewalls -- SSH

One of the most popular ways to remotely administer a Linux system is using SSH. SSH provides an encrypted command line session from any remote machine. It's the secure replacement for the once ubiquitous *telnet* application. These days using telnet is frowned upon since it sends all data unencrypted in clear text. This step is totally optional, but if you want to work on your machine remotely, this is one of the best methods. First, install the ssh package:

Firewall:~# apt-get install ssh

The Debian package will prompt you for a few options. Select *Yes* to all three options: allow SSH version 2 only, run setuid, and install the server.

Our configuration file for this daemon is /etc/ssh/sshd_config

Firewall:~# nano -w /etc/ssh/sshd_config

Most of the defaults should be fine here. Personally I like to change *PasswordAuthentication* to yes, assuming you want to log in using your account password. Read through the configuration files, and change anything you want.

Depending on the firewall policy you set up earlier, you may or may not need to make adjustments. If you used the example policy of accepting all local traffic to anywhere, then SSH should be accessible from any local machine. You can also allow access from the internet by creating an instance in the /etc/shorewall/rules file, if you're feeling brave. Finally, if you want to tighten security, you can adjust the rules to only allow specific local machines to access the SSH port. This is great in an office network where only the administrator's workstation should have access.

Proceed to Step 10 - NTP

Debian Firewalls -- NTP

NTP is a handy little service to have running. Also known as the Network Time Protocol, it will keep both your firewall's clock synchronized, and allow any clients on your LAN to synchronize their clocks with the firewall:

Firewall:~# apt-get install ntp-server

The configuration file is /etc/ntp.conf.

Firewall:~# nano -w /etc/ntp.conf

The default settings should be ok. If you choose to synchronize your entire network to the firewall, you will probably want to add *option time-servers* to your dhcp configuration file. Set the server address as your firewall's internal address.

Proceed to Step 11 - Proxying

Debian Firewalls -- Proxy

Proxies can fulfill several different functions on network gateways. Two of the most popular are increasing speed, and increasing security and control. A proxy acts as a "middle man" between the client PC and the internet server. When a client requests a connection to a source outside the gateway, it sends the request to the proxy. The proxy examines the request, and forwards it to the server. The proxy then receives the response, and performs the appropriate actions before passing it on to the original host. A caching proxy will store the data, allowing it to be retrieved very quickly the next time it is requested. A security based proxy will examine the outgoing and incoming traffic for malevolent data and either block it or sanitize it.

One of the most popular Linux proxy programs is Squid (http://www.squid-cache.org). We'll be using this as our caching proxy. Later, we'll check out Privoxy (http://www.privoxy.org), which is a lighter solution designed to act as a security proxy.

System Requirements Notice: Toward the beginning of the tutorial series we noted how well a small obsolete computer can run a Linux gateway. While it remains true that a proxy will technically run fine on any small system, proxies often demand larger systems to really perform well. A caching proxy needs a lot of space to effectively store all the data that it requests from the internet. Any proxy needs RAM and processing power to parse all that data mid-stream. That said, if you really want to benefit from a proxy you'll want bigger hard disks, more RAM, and faster processing than a bare-bones router. How much you need is ultimately up to you and how you plan on using your new proxy.

First, we install Squid:

```
Firewall:~# apt-get install squid
```

Alternative Alert! - Squid
Proxying is anything but a small field, so of course there are a lot of alternatives. I personally recommend checking out Privoxy (covered below) if you're looking for security vs. caching. In both fields, there are plenty of alternatives.

Our configuration file for this daemon is /etc/squid/squid.conf

```
Firewall:~# nano -w /etc/squid/squid.conf
```

As with a lot of our services, proxies are powerful tools, so there's lots you *can* configure. As always, I recommend reading the configuration, and in this case check out the FAQ on the Squid website as well.

In this case, we're just going to change the listening interface and port. By default Squid listens on all its interfaces, on port 3128. We're going to change this to listen on only the internal interface, on the standard proxy port of 8080. To do this, uncomment and change the http_port parameter.

http_port 192.168.1.1:8080

Squid supports a lot of authentication options and access control lists (acls). These are important when your proxy is exposed to untrusted clients. For this example, we'll be running only on our trusted home network, so there's no need to restrict access. By default, Squid only accepts connections from the local machine. To change this, we have to tell it to accept all connections from local IP addresses:

Find this line in the configuration:

acl localhost src 127.0.0.1/255.255.255.255

and change it to this:

acl localhost src 192.168.1.0/255.255.255.0

This will allow any client on the 192.168.1.0 network to access the proxy. Only do this if you trust all your local machines. If you do not, read the comments and adjust the ACL to restrict access to certain machines only!

That's all we need. Save the file and exit, then run:

Firewall:~# /etc/init.d/squid restart

Now all you have to do is change the proxy settings for the web browser on your PC. Change the HTTP proxy host to 192.168.1.1, and the port to 8080 (assuming you change it as we did above). The next time you request a website, the browser will query the proxy rather than the website itself. If the process fails, check the log files in */var/log/squid/* (any errors should be printed there).

Alternative Alert! - HTTP Proxy
We just configured a passive proxy...you have to manually configure each web browser to access it. Squid can also be configured as a transparent proxy. Transparent proxies run on the standard HTTP port (80) and intercept all traffic to the web, without needing the browser to be specially configured. Transparent proxying is out of the scope of this article, but if you read the configuration files you can probably figure it out fairly easily.

Now that we've covered content caching, lets take a quick look at privacy caching with Privoxy. Keep in mind these are totally separate solutions and you can choose to implement one, both, or none, depending on your needs.

Firewall:~# apt-get install privoxy

Privoxy uses several configuration files in /etc/privoxy and you should explore all of them. */etc/privoxy/config* provides the general program options. Open this file, and change *listen-address* to the address of your LAN interface:

listen-address 192.168.1.1:8118

Now restart Privoxy:

Firewall:~# /etc/init.d/privoxy restart

This is all you need for the base configuration. Direct your browser to the new proxy port (8118) and Privoxy will kick in. Privoxy is configured to protect your browsing privacy in several ways by default, including filtering certain fields out of your HTTP requests, and by blocking unwanted data from some much maligned advertising websites. For full details, check out the config files and http://www.privoxy.org. Of course, all of this is highly customizable. Check out the filters and customize them to meet your needs. Privoxy also has a cool web-based administration interface should you choose to enable it.

Proceed to Step 12 - Dynamic DNS

Debian Firewalls -- Dynamic DNS

Most hosts running internet based services use DNS to make their machines easily accessible by name. If you're running some type of server behind your firewall, you'll probably want to give it a domain name (www.mysite.com) to make it easily accessible. Domain names can also be handy for personal use, like if you need to connect to your machine remotely (with PPTP for example). For we humans, names are just simply easier to remember than numbers.

A dedicated server usually runs DNS with a static IP address. It would be a problem if Google's IP address changed constantly, wouldn't it? Unfortunately for those of us running personal servers on basic broadband connections, we have dynamic IP addresses, which are designed to change every so often. To compensate for this, we can use Dynamic DNS. The principle is simple: an extra service runs on the host machine and monitors the IP Address. When it changes, the Dynamic DNS service automatically takes notice, and updates the DNS record on the DNS server. This keeps downtime to a minimum and eliminates the need for you to constantly be updating your records. Best of all, it's free and easy to set up!

Lots of companies offer free Dynamic DNS hosting. For this tutorial we'll be using the popular DynDNS service. DynDNS offers many DNS services, including a reliable and FREE dynamic DNS service.

Alternative Alert! - DynDNS
DynDNS is only one possibility. Just Google "dynamic dns" for lots more! Best of all, the free dynamic dns client we're covering supports just about any of them!

To start off, you'll need to register an account with any internet based dynamic dns service. If you choose DynDNS, you can sign up here: http://www.dyndns.com/services/dns/dyndns/. Registration should be simple and painless, just sign up with any username and password. **Make sure not to use a password you've used for other accounts! More on this later...** Once you've created your account, you need to add a host record to your account. You'll have to name your new site, most likely it will be in the form of [mysite].dyndns.org. That should be all the account configuration necessary.

The program we'll be using as our client is called ez-ipupdate. It's a very light dynamic dns client written in c that supports lots of different DNS services. I'll spare you another Alternative Alert, but just know there are a lot of other clients you could choose if you want.

```
Firewall:~# apt-get install ez-ipupdate
```

ez-ipupdate comes with lots of example configurations for various dynamic dns services. To view them, list the files in */usr/share/doc/ez-ipupdate/examples/*:

```
Firewall:~# ls -al /usr/share/doc/ez-ipupdate/examples/
```

The actual examples are stored in */etc/ez-ipupdate*. To start, copy one of the example configs there, or just create your own:

```
Firewall:~# cp /usr/share/doc/ez-ipupdate/examples/example-dyndns.conf /etc/ez-ipupdate/ez-ipupdate.conf
Firewall:~# nano -w /etc/ez-ipupdate/ez-ipupdate.conf
```

Now we just need to give the program some basic information. Here's the entire config first:

```
#!/usr/sbin/ez-ipupdate -c
```

service-type=dyndns
user=MyUserName:MyPassword
host=myhostname.dyndns.org
interface=eth0
max-interval=2073600
run-as-user=ez-ipupd
cache-file=/var/cache/ez-ipupdate/default-cache
daemon

Now lets break it down: The first line is necessary because this file will actually be used as an executable. *service-type=dyndns* defines the type of service we're working with, in my case dyndns (see other files in the examples directory for more). *user=MyUserName:MyPassword* tells ez-ipupdate how to log into the dynamic dns service. This should be the account information you used when you signed up. This is also why you should use a unique password...it will be stored and transmitted in clear text! *host=myhostname.dyndns.org* should match the host name you chose on with the dynamic dns service. You can have more than one per account, so you need to specify which one you're updating. *interface* specifies the interface with the IP address you're tracking...make sure to use your ISP interface! *max-interval* controls how often updates must be made...don't make it too small or your service provider may not be happy. *run-as-user=ez-ipupd* is a security feature telling the program not to run as a privileged user. Finally, *daemon* tells the service to run in the background. Uncomment this for daily use.

Now we can start the service to test it:

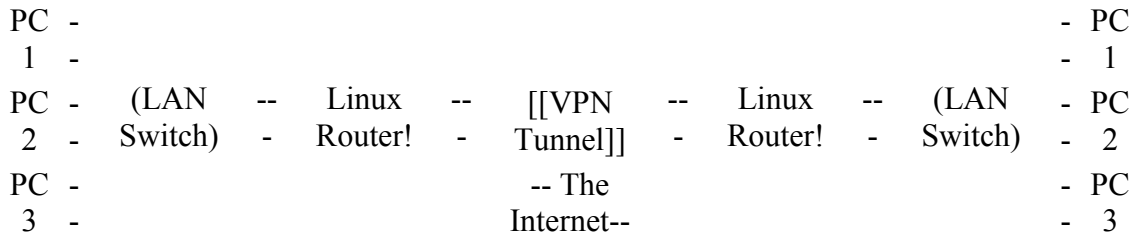
Firewall:~# /etc/init.d/ez-ipupdate start

That should do it. Check /var/log/syslog for the program messages. It will tell you if you've successfully updated your record or if there was an error that needs fixing. Your IP will now (soon) be accessible via your domain name, anywhere in the world! One important note: DNS records take some time to propagate globally. Even if your client works perfectly it may take an hour or two to actually see changes around the internet. Sometimes you just have to be patient.

Proceed to Step 13 - IPSec VPN

Debian Firewalls -- IPSec

In this section we will work toward creating a basic site-to-site VPN using IPSec. First, we should get a few technical points out of the way. IPSec can be used in multiple arrangements: the most common are site-to-site and access. What we'll be creating here is the site-to-site version. In essence, we'll be bridging two remote locations transparently using the internet. All the VPN work is done by the firewall computer...the host computers on the LAN don't need to do anything to participate, it's completely transparent. That said, for this to work you will obviously need two firewalls at different locations, both capable of IPSec VPN traffic. In this example, we will assume we have a remote location that's using a Linux firewall just as we have at our original location. In other words, it will be a mirrored version of our original network diagram, with the internet in the middle:



An important note on topology: The remote network must be on a different subnet than the local network. For example, you can not have both networks using 192.168.1.0/24.

Now hopefully, when you consider that you're about to attempt sending local traffic over the public internet, the question of security will come to mind. After all, you don't want everybody watching your LAN traffic, do you? Well fear not, IPSec was designed with security in mind...the fact that it's short for IP Security should tell you that. :) In fact, major corporations use this same technology every day to transport data between remote business locations. And since Linux is so versatile, we'll be using some of the most powerful encryption on the market today (at no additional cost!). Another cool fact is since we're using standard IPSec traffic your firewall *should* be interoperable with IPSec devices from any number of vendors.

Alternative Alert! - IPSec
A number of readers have pointed out to me that a competing product, known as OpenVPN, is also available for creating VPNs. I chose IPSec for a couple reasons: First of all, I've never used OpenVPN, so it might be tough writing a tutorial about it! Even so, I think IPSec is implemented so widely that it's an important topic for any would be net-admin to cover. Even though the implementation differs, the core fundamentals are the same whether it's a simple Linux box or a high-end VPN concentrator. On the other hand, I don't want to discourage anyone from trying OpenVPN...In fact I always encourage branching out!

First we need to install a couple IPSec packages. *Racoon* is responsible for the key exchange portion of our IPSec connections, and will be used to configure our kernel with all the necessary IPSec information. The *ipsec-tools* package provides just what it sounds like...supporting tools for our IPSec implementation.

Firewall:~# apt-get install racoon ipsec-tools

You will now be prompted with two options: direct configuration or using the racoon-tool program.

We'll be using racoon-tool which nicely automates most of the configuration needed to make our IPSec connection. I recommend everyone start with racoon-tool. If you want to learn the lower-level configuration, just look at the files generated after we run racoon-tool, and you will know what the "direct configuration" files will look like. Most likely on a non-Debian system you would just be using the direct configuration method.

Now that our program is installed, we'll have to tackle the configuration. Since we're using racoon-tool, we can do this from a single file: */etc/racoon/racoon-tool.conf*

Firewall:~# nano -w /etc/racoon/racoon-tool.conf

You will see that there are nice example configurations already commented into the config file. This file will need to be edited on both the local and remote networks, assuming you're setting both up at once. Keep in mind that **all settings must match** on both sides! If they don't, IPSec won't be able to negotiate a connection.

For the remainder of the configuration examples we'll pretend the local network has the following properties:
ISP (external) address: 10.1.1.1
LAN (internal) address: 192.168.1.1 (same as always)

For the remainder of the configuration examples we'll pretend the remote network has the following properties:
ISP (external) address: 10.100.100.1
LAN (internal) address: 192.168.100.1

Let's show the entire racoon-tool.conf configurations first, then break them down:

Local Configuration	Remote Configuration
connection(ToRemote): *src_range: 192.168.1.0/24* *dst_range: 192.168.100.0/24* *src_ip: 10.1.1.1* *dst_ip: 10.100.100.1* *authentication_algorithm: hmac_sha1* *admin_status: yes* *peer(10.100.100.1):* *passive: off*	*connection(ToLocal):* *src_range: 192.168.100.0/24* *dst_range: 192.168.1.0/24* *src_ip: 10.100.100.1* *dst_ip: 10.1.1.1* *authentication_algorithm: hmac_sha1* *admin_status: yes* *peer(10.1.1.1):* *passive: off*

verify_identifier: on *lifetime: time 30 min* *hash_algorithm[0]: sha1* *encryption_algorithm[0]: aes* *my_identifier: address 10.1.1.1* *peers_identifier: address 10.100.100.1*	*verify_identifier: on* *lifetime: time 30 min* *hash_algorithm[0]: sha1* *encryption_algorithm[0]: aes* *my_identifier: address 10.100.100.1* *peers_identifier: address 10.1.1.1*

As you can see, the only settings that vary are the IP addresses configured on each side. Each configuration file also has roughly two parts, the connection entry and the peer entry. The connection entry defines which peers are involved, which the peer entry defines specific settings for the peer. Keep in mind that you don't need to limit yourself to one of each; you can potentially run many different IPSec connections and peers just by adding additional entries. Now we'll explain the configuration:

connection(ToRemote):	Defines a connection. You can have as many of these as you want. The name in () is arbitrary.
src_range: 192.168.1.0/24	The range of IP addresses on the local LAN (internal network). Must be in a separate range from the remote network.
dst_range: 192.168.100.0/24	The range of IP addresses on the remote LAN (internal network). Must be in a separate range from the local network.
src_ip: 10.1.1.1	The ISP (external) IP address of our local firewall.
dst_ip: 10.100.100.1	The ISP (external) IP address of our remote firewall.
authentication_algorithm: hmac_sha1	Defines hmac_sha1 as the algorithm used for authentication. Options include hmac_sha1 and hmac_md5. SHA1 is generally considered stronger.
peer(10.100.100.1):	Defines a peer configuration. The IP address should match the dst_ip defined in the connection.
passive: off	If passive is on, the router won't initiate the connection. With passive off, a connection will be attempted.
verify_identifier: on	Verify the identity of your remote endpoint (probably a good idea).
lifetime: time 30 min	The lifetime after which the connection must be renegotiated. A shorter lifetime gives malicious parties less time to crack into a given session.

hash_algorithm[0]: sha1	Hashing is used to verify all traffic has not been tampered with en-route. Each packet is hashed as it leaves, and hashed again when it arrives at the remote endpoint. If the hashes don't match, the packet has been altered and is dropped. MD5 and SHA1 are available. You can also specify more than one.
encryption_algorithm[0]: aes	Here you specify the encryption algorithm used. This is what keeps your data unreadable by 3rd parties as it travels over public networks. You should have lots of options for this. I've chosen AES as it is one of the stronger types available. Options include aes, des, 3des, blowfish, cast128. You can also specify more than one.
my_identifier: address 10.1.1.1	How you identify yourself to the remote host. Can include IP Addresses, domain names, and others.
peers_identifier: address 10.100.100.1	How to identify the peer (same opts as above).

There are multiple options for how to authenticate the two peers with each other. The most secure (and also most complex) are to use certificate files. The simpler option (but less secure) is to use a pre-shared key. We'll be using a preshared key here. I recommend generating a long string of random characters to use as your key. **Do not use words or short keys as it's a high security risk!**

By default, racoon-tool configures racoon to use the pre-shared key file */etc/racoon/psk.txt*:

```
Firewall:~# nano -w /etc/racoon/psk.txt
```

The format of the file is very simple, the IP address of the remote peer's ISP (external) interface, followed by the key. The key should match between the two peers (and ONLY between two peers!).

On the remote firewall:
10.1.1.1 G9KuwsuiuBHGu92S7AC4eKHt5dnpL8548Q46Rdiuke9zCb8U

On the local firewall:
10.100.100.1 G9KuwsuiuBHGu92S7AC4eKHt5dnpL8548Q46Rdiuke9zCb8U

That's the end of our racoon configuration. Now we have to "reload" the tool. Racoon-tool will parse the config, and generate configuration files for the underlying IPSec apps:

```
Firewall:~# /etc/init.d/racoon reload
```

If you've made any invalid entries, racoon-tool should tell you. Even though IPSec is now configured, we can't quite start our connection yet. First we have to configure our firewall to allow the IPSec traffic through.

First we have to add a new zone called "VPN" to shorewall:

Firewall:~# nano -w /etc/shorewall/zones

vpn VPN VPN to remote network

Now we have to define an IPSec tunnel using the */etc/shorewall/tunnels* file. Remember to use the remote IP address as the gateway:

Firewall:~# nano -w /etc/shorewall/tunnels

ipsec net 10.100.100.1

Now we have to define the *vpn* zone as a sub-section of the *net* zone (since they use the same physical interface on the firewall). We do this by identifying the interface the VPN traffic comes in on (our ISP interface), and the subnet the remote network is using. This way when the firewall sees traffic coming in on our ISP interface using the remote LAN subnet, we classify it as part of the VPN zone.

Firewall:~# nano -w /etc/shorewall/hosts

vpn eth0:192.168.100.1

We also need to exclude our remote subnet from being translated as NAT/PAT traffic as it leaves the external interface. The VPN creates an exception where we want our internal IP addresses to remain *intact* as they leave the firewall, rather than being converted to external addresses. We only have to add on to our original masq config, this is not a new line:

Firewall:~# nano -w /etc/shorewall/masq

eth0 eth1

becomes

eth0:!192.168.100.0/24 eth1

Last but not least, our policy needs to account for the VPN traffic. As always, you may choose to permit all using the policy, or to restrict the policy and make specific rules for the VPN zone in the rules file. In a corporate environment, a restrictive policy remains best...perhaps you only want to permit access to a specific server over the VPN. In a home environment, you may wish to just permit all VPN traffic. Make sure this setting goes *before* the *net* and *all* statements in the policy.

Firewall:~# nano -w /etc/shorewall/policy

vpn all ACCEPT

That should be the final configuration! Now we have to restart shorewall to take the new configs, followed by starting racoon to begin our IPSec connection:

Firewall:~# /etc/init.d/shorewall restart
Firewall:~# /etc/init.d/racoon start

The best way to test is to send some simple traffic between the local and remote LANs. Try pinging a remote computer or accessing a remote service. Since

there was a lot of room for error here, there's a good chance you may have a problem. If traffic fails to cross successfully, there are two good places to search for errors. /var/log/syslog will show you any errors with racoon. Check closely for errors negotiating the IPSec link. Also, check the firewall logs for dropped traffic from the remote network. If you see this, the firewall is still blocking your incoming VPN traffic and needs to be adjusted.

One last important note: Traffic that relies on broadcast will **not work** over the VPN link. Typing "ping [computer name]" will fail; you must use "ping [ip address]". This is because broadcast traffic only works within your local subnet, and the remote VPN is on a different subnet. To be able to use hostnames across the VPN link you will need to use a local DNS or WINS server to resolve the names.

Proceed to Step 14 - Intrusion Detection (IDS)

Debian Firewalls -- IDS

An Intrusion Detection System (IDS) is a security application or appliance that can be deployed at different levels in a network. A host-based IDS resides on a network endpoint such as a workstation, and attempts to detect malicious activity on the machine. Alternatively, a network-based IDS is deployed transparently on a network and listens to the network traffic for malicious packets. IDS can use different methods of detecting problems, but the most basic method is using signatures. Like an antivirus program checks files for virus signatures, and IDS will check network traffic for patterns associated with malicious network activity.

By far the most popular and powerful of the free IDS solutions is Snort (http://www.snort.org). It's important to note that Snort is not a firewall. Our Snort configuration is intended to only detect problem traffic, not block it (although with some work it can be configured otherwise - see inline mode in the Snort manual).

Configuration Notice: While all the examples in this tutorial are designed to be rather basic, the Snort configuration here is very minimal. The idea is, as always, to let you get up and running and allow you to research and expand the configuration to suit your needs later on. The way an IDS is useful will be different for everyone based on their needs, so it's up to you to identify which detections you need and which you don't. I HIGHLY recommend everyone attempting this step in the guide visit Snort.org for two reasons:

- Documentation: Learn how to take advantage of the multitude of features built into Snort. The documentation is long, but valuable.
- Updates: The Debian package comes with a pre-defined rule set, but the rules are updated regularly. The community rules are free for download and will allow your IDS to detect the latest threats.

To start off, we need to install the snort package:

Firewall:~# apt-get install snort

Debian will do some Snort config for you at run time.

You will be prompted to select an interface for Snort to listen on. We'll be configuring it on our internet interface, since that's where we expect the threats to come from. It can also be run on the internal interface to inspect outgoing traffic.

You have to set the IP address range of your home network. In our example, this will be the 192.168.1.0/24 subnet.

A script will mail IDS statistics to the specified user(s). Make sure they have some way of receiving the information.

Snort has several configuration files to be aware of. Some of the most important are:

- */etc/snort/snort.conf*: This is the primary configuration file, and deserves a read through and usually some tweaking.
- */etc/snort/snort.debian.conf*: This file is created by the Apt installer. Your initial options are placed here leaving snort.conf unchanged.
- */etc/snort/threshold.conf*: This file lets you rate-limit alerts if you're seeing a lot of the same warning. It also lets you supress warnings completely.
- */etc/snort/rules/*: This directory contains the snort rules (signatures). These can be manually updated with new rules from Snort.org, or edited by hand to provide new detections.

When a new detection is made, it will be logged in /var/log/snort/alert by default. These logs aren't very pretty, but you can find a lot of log interpreters available to supplement the installation. When Snort starts, it will give a lot of information in /var/log/syslog about its configuration, what is enabled, and what isn't.

Now let's do some Snort tweaking to match our network! Keep in mind that this is just a sampling to get you started. We won't attempt to cover all the included features, that's up to you.

Firewall:~# nano -w /etc/snort/snort.conf

The VAR_* options at the beginning allow you to limit what addresses are scanned for certain types of vulnerabilities. For example, you can set it to only scan traffic to your web server at 192.168.1.10 for HTTP vulnerabilities:

var HTTP_SERVERS 192.168.1.10/32

The middle of the config contains the preprocessor configs. The preprocessors are Snort modules that run before the actual snort detection engine, and can extend Snort's core capabilities with features like port scan detection, http traffic inspection, ip defragmentation, and much more. You should read through each and configure it to your needs...the Snort website documents all the options well.

At the bottom you will find lots of file includes. The rulesets are included, as well as some additional configuration files. If you want to exclude a certain rule set from being processed, simply comment it out:

#include $RULE_PATH/tftp.rules

Keep in mind if you only want to avoid a single rule, rather than the entire set, you should use the threshold.conf file to suppress it by signature. Once you're happy with the configuration, sit back and watch everything work. Don't forget to monitor your new IDS for unexpected results...high traffic networks can quickly generate a lot of detections if not configured properly.

Proceed to Step 15 - Diagnostics with IFTop & TCPDump

▶▶Debian Firewalls -- Diagnostics with IFTop & TCPDump ◀◀

One of the fun parts about building your own firewall is having the ability to use diagnostic tools to monitor your traffic. Here we'll take a quick look at two tools, IFTop and TCPDump, which are quite useful for troubleshooting.

Linux and Unix users are probably familiar with the ubiquitous *top* program which monitors processes. IFTop (http://www.ex-parrot.com/~pdw/iftop/) is a network-centric take on the *top* program which lets you monitor network connections in and out of your machine.

Incase you haven't noticed the running theme, all the packages we've used so far are available in Apt. To start off, let's install IFTop:

Firewall:~# apt-get install iftop

There's no configuration necessary, just run *iftop* from the command line.

Firewall:~# iftop

IFTop will display a list of connections on your console/terminal screen, showing the two machines involved in each connection, and the bandwidth for each. Commands are entered with single keys: Hit 'h' for the help screen to view available settings. When you're finished, hit 'q' to return to the command line.

TCPDump (http://www.tcpdump.org) is the premiere packet sniffer of the 'nix world. An extremely powerful (and potentially complex) tool, TCPDump can monitor, capture, and decode all packets crossing a given interface. While not very fun to watch, it is indispensable when it comes to troubleshooting connections.

Firewall:~# apt-get install tcpdump

TCPDump is generally configured from the command line at run time. There are **far** too many commands to cover here, so check out the manpage and online documentation. Here's a few quick examples:

Capture all traffic involving host 192.168.1.20:

Firewall:~# tcpdump host 192.168.1.20

Capture HTTP packets on eth0:

Firewall:~# tcpdump -i eth0 port 80 and tcp

Capture HTTP packets on eth0, and decode packets in ASCII:

Firewall:~# tcpdump -Ai eth0 port 80 and tcp

Capture packets on eth0 on port 80 or 8080:

Firewall:~# tcpdump -i eth0 port 80 or port 8080

Capture UDP packets to port 53 (DNS) originating from 192.168.1.20:

Firewall:~# tcpdump dst port 53 and udp and src host 192.168.1.20

Dump the packets to a file instead of the screen...:

```
Firewall:~# tcpdump -w dump.txt host 192.168.1.20
```

...then read in the packets to process later:

```
Firewall:~# tcpdump -r dump.txt
```

Proceed to Step 16 - Custom Kernels

Debian Firewalls -- Custom Kernels

The Linux kernel is the core of the Linux operating system. The Kernel controls all the central system functions and is the heart of your firewall system. Each distribution comes with its own specially designed version of the kernel, based on the feature set the designers wish to offer. Unlike many commercial operating systems however, the kernel source itself is freely available, and you can download and create your own specialized versions.

Why should you build a custom kernel?

- The kernel developers are a great group that work tirelessly improving the operating system. Each new version brings literally thousands of bug fixes and enhancements that your system can benefit from.
- Streamline your system! When building a specialized system like a firewall, there are a lot of features your operating system doesn't need to support. Why waste memory and clock cycles running an operating system with multimedia and peripheral support that it will never use? Instead, a custom kernel will be extra efficient by supporting only the necessary hardware and features, allowing you to make the most of your resources.
- Patch in new features: lots of development is done outside the official kernel. Building your own allows you to add external patches for non-official features.
- Cutting edge technology: most distributions (including Debian) stick with the same kernel version for a prolonged period of time. This isn't a bad thing if you're looking for stability and support, but if you like the latest and greatest, the best way to get it is straight from the (kernel) source.

Why should you *not* build a custom kernel?

- The distribution kernels are usually expertly planned to be stable. While kernel developers do a great job fixing bugs, the cutting edge kernel releases are far from perfect. If you're not willing to accept some occasional bugs and down time, stick with a more mature and professional kernel build.
- If you're really new to Linux. Of course, you can't learn unless you try! And with proper caution, this is all perfectly safe. However if you have no previous Linux experience, you may be well advised to spend a little time poking around with the generic system before you dive too deep.
- If you're not familiar with the hardware and software already in your system. Customizing the kernel requires you pick and choose what features your system will support. If you don't know what elements you need, you won't have much of a chance designing a functional kernel.

You can always retrieve the latest source from the kernel website: http://www.kernel.org. The 2.6 branch is the latest, while the 2.4 branch is the previous series. I recommend 2.6 for all new systems. I also recommend running the "stable" releases, unless you really know what you're doing and are willing to report bugs to the developers. As of the time of this writing 2.6.15 is the current stable release.

First, we need to install some software that will be necessary to compile the kernel. Bzip2 is needed to unpack the source. libncurses5-dev is needed by the configuration menus. GCC and make are needed to compile the source.

```
Firewall:~# apt-get install bzip2
Firewall:~# apt-get install libncurses5-dev
```

Firewall:~# apt-get install gcc
Firewall:~# apt-get install make

Now we download the full source package (the "F" link on the source page). You should change the URL below to match the address for the current stable source package.

Firewall:~# cd /usr/src
Firewall:/usr/src# wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.tar.bz2

Unpack the source (using the appropriate filename):

Firewall:/usr/src# tar -xjf linux-2.6.15.tar.bz2

Change into the source directory:

Firewall:/usr/src# cd linux-2.6.15

The kernel configuration is stored in the .config file. Luckily, the kernel comes with a handy text-based configuration menu which makes navigating the hundreds of options much simpler. To enter the menus, run a "make menuconfig":

Firewall:/usr/src/linux-2.6.15# make menuconfig

Now you should be in a text-based menu. What you do from here is largely up to you. There's little point in attempting to cover all the options, as everyone will want a different configuration. Instead, here are some important guidelines and tips:

- There are two ways to include a feature in the kernel. All items can be built directly into the kernel by placing an asterisk next to the item [*]. Items preceded with angled brackets <> can also be built as modules. Modules can be dynamically loaded and unloaded, and are the preferred method of building device drivers and optional system components. To build as a module, place an <M> next to the item.
- When you wish to exclude an item from your kernel, simply clear the brackets next to it.
- Certain critical system features must be built directly into the kernel (not as modules)! Do not build support for any devices needed to boot the system as modules. This includes (but is not limited to), the motherboard, processor, IDE devices (hard drives), and the hard drive filesystems. If you use PCI controller cards for your drives, they need to be included too. Examples of optional system devices that can be built as modules are sound cards, network cards, video cards, firewall components, and any peripherals (printers, etc).
- By default, Debian 3.1 uses the ext3 filesystem for the system disks. This means you must build ext3 support directly into the kernel, or it will not boot. This is done by placing an asterisk [*] next to the ext3 option. Of course if you chose a different filesystem at install time, you should select the appropriate filesystem support in your kernel.
- Do not uninstall the kernel that came with your distribution. Customizing kernels is prone to error and oversight, even if you've done it before. Always keep one known working kernel installed at all times. That way, if you make a mistake or find a bug and your new kernel won't boot, you can revert to the old kernel straight from the boot menu.
- Try working in steps. If you make major changes, it can be hard to track down an error. Start with a lot of features enabled and remove only a few functions at a time, so you can easily backtrack to a working version.
- Keep backup copies of your configuration files. Once you get a working kernel, simply copy the .config file to a safe place, and you can copy it

back whenever you need it.

- The menuconfig includes help documentation for most of the items. If you're not sure what an entry does, try typing an ? to access the help entry. If all else fails, Google it!
- You can easily tell what modules your original system is running by executing the "lsmod" program from the command line. This will, as the name suggests, list all the currently loaded modules. Keep in mind, this won't tell you which options are built directly into the stock kernel.
- The kernel doesn't just support hardware, software functions can be affected as well. For example, make sure to leave some iptables/netfilter options enabled...if you disable them your firewall apps won't run!

When you finish, exit the menu and save your configuration. You are now ready to compile the new kernel. This is done simply by running "make":

Firewall:/usr/src/linux-2.6.15# make

The kernel will now build. If you're using an old machine, this can take awhile (an hour or two, depending on how large of a kernel you're building). Once it finishes, first install the new modules:

Firewall:/usr/src/linux-2.6.15# make modules_install

Finally, install the kernel itself:

Firewall:/usr/src/linux-2.6.15# make install

Your new kernel has now been copied to the /boot directory and your modules are now in the /lib/modules directory. The final step is to update your bootloader. Debian uses the grub bootloader, and comes with a handy script that will take care of this for you:

Firewall:/usr/src/linux-2.6.15# update-grub

That’s it! It’s really that simple. Now just reboot your system, and choose the new kernel from the boot screen. If you get an error during booting, try to make note of the error message. Then select your original kernel from the boot prompt, and try again.

When you finish, you can save disk space by deleting the compiled binaries from the source directory. You should also do this after changing the configuration, before re-compiling the kernel.

Firewall:/usr/src/linux-2.6.15# make clean

Kernel updates come out frequently. When you wish to upgrade, download and extract the new source. Copy the .config into the new source directory. Then run "make oldconfig" against the new source:

Firewall:/usr/src/linux-2.6.15# make oldconfig

This command will prompt you for any new, relevant options that may have appeared in the new source, and will make sure your old configuration is appropriate for the new version. After this, simply make and install again.

Back to the Table of Contents

Q: A Linux router sounds great, but I don't want to build my own from scratch. Are there any prebuilt firewall systems available?

A: Yes! This is not exactly a new field and there are lots of professionally developed options. Among the most notable are Smoothwall: http://www.smoothwall.org, IPCop: http://www.ipcop.org, and ClarkConnect: http://www.clarkconnect.org. All of these are quality Linux-based operating systems designed for use as network firewalls/routers/gateways.

Q: I have a wireless firewall and I don't want to give up my wireless for a Linux router! What can I do?

A: Easy, keep it! I currently run a Linksys Wireless G firewall/router behind my Linux firewall as a wireless access point. They key is to just ignore the firewalling capabilities of the wireless routers. Turn off it's DHCP and firewall services via the management interface. The, connect the gateway and your host machines on the general switching ports...this allows your wireless devices to continue functioning as a wireless access point and leaves the power lifting to your Linux box.

Q: So what is the advantage of those commercial hardware firewalls that cost so much more?

A: As I point out in the introduction, Linux firewalls have lots of benefits...power, flexibility, and low cost at the top of the list. Just like everything else in computing, however, it's not a perfect solution.

Most hardware firewalls perform at least some, if not most, of their functions using ASICs (Application Specific Integrated Circuits). ASICs perform system operations in physical circuits rather than in software, making it possible to process data at raw hardware speed. When you're running an internet backbone or the core routing for a major corporation, this kind of raw speed is important. When you're running a home or small office connection via broadband, or even an average LAN connection, you will never reach the kind of network speeds where you'll notice a difference. I've maxed out the LAN connection at my college residence without noticing a performance hit on my Linux gateway.

In a similar way, the software involved is also an issue. Linux is basically a general purpose operating system, meaning even with customization you've got at bare minimum application software running on top of an operating system, which is a potentially unnecessary layer of abstraction. A dedicated network device on the other hand may run software designed specifically for that device. This often improves stability and performance since the entire OS is designed specifically for the hardware. Don't be fooled though...many consumer devices, such as the Linksys WRT54G series wireless firewalls...run a Unix derived operating system under the hood. I personally run custom Linux kernels on all my routers to maximize speed and efficiency and eliminate overhead.

Q: How about providing the guide in a single file format?

A: Now, by popular demand, a PDF version is available! Keep in mind that the quality may not be as good as the actual web version, but it's better than nothing. :)

Q: Can one of these firewalls be run with a GUI? Why don't you add a GUI to your installation?

A: GUIs are wonderful for workstations, but as any Linux/Unix administrator worth their salt will tell you, a GUI doesn't really belong on a network device. This is a very common topic in server administration, and it applies to firewalls and routers too. Of course you *can* run any GUI you want on your system. But when it comes down to it, a firewall (or server) is not meant to be interacted with on a regular basis. After configuration, they usually just facelessly perform their given functions. So for that 98% of the time that no one is using the user interface, it's doing nothing more than wasting memory

and sucking up processor cycles better used elsewhere.

Q: What about webmin?

A: Same principle as above, basically. You really don't need a graphical interface to get the job done, and the servers required to run it just waste computing power and resources. Not to mention the fact that if you intend to use it remotely over the internet, you may be exposing yourself to a serious security threat! So of course, use it if you want it, just know the possible consequences.

Q: Is this a stateful firewall?

A: Yes, starting with Linux kernel 2.4 the netfilter/iptables package has provided stateful packet filtering capabilities. Since this tutorial assumes that iptables is in use with either kernel 2.4 or 2.6, your firewall will have full stateful packet filtering ability.

Back to the Table of Contents