# Home Networking In Linux

## Iptables Firewall, Routing, Wireless, and More

Scott Paul Robertson
`http://spr.mahonri5.net`
`spr@mahonri5.net`

December 10, 2006

## Why Build My Own Router?

- With most ISPs, you only get a single uplink jack, but you probably have more than just the one computer. Most people solve this by purchasing a all-in-one router from one of the major companies: Linksys, Belkin, Netgear, DLink, etc. Usually you configure them through some annoying web-based utility.
- In my experience with these, you either get lucky and everything always works, or you're hitting the reset button about everyday to reboot the thing so everyone can get on the wireless network. We won't even begin to get into the evil that is the configuration utility.
- For all these reasons, we want a better, more flexible, more configurable solution.

## What Do I Need?

- You will need a machine that runs Linux. For very simple setups a Pentium 1 should work, but the more you want to do (security for wireless, web filtering, complex firewall rules) the faster your processor should be. 400 MHz is sufficient for the more complex setups.
- You will need to be comfortable running a system *without* X, familiar with kernel settings, and some familiarity with a scripting language.
- You will need to pick a distro that can handle running on a slower machine. SuSE, Fedora, Gentoo, and Unbuntu are probably bad choices. I recommend Debian.
- You'll probably want to do a minimum install.

## Kernel Configuration

- After installing Linux onto the box, you'll want to be sure that your kernel has a few standard options set.
- Under `Networking--->Networking options` you'll want to enable:
  - IP: TCP syncookie support
  - Network Packet filtering
- Under `Network Packet filtering--->Core Netfilter Configuration`
  - Netfilter Xtables support
  - MARK, CLASSIFY, NFQUEUE, comment, conntrack, limit, max, state
- Under `IP: Netfilter Configuration`
  - Connection tracking: FTP, IRC
  - IP tables support: range, multiple port, TOS, recent, packet filtering, LOG, NAT (MASQUERADE, REDIRECT), mangling

## Basic Networking

- Usually, an ISP will provide us with a single address to use when connecting to their network. To have multiple computers go through the one address we need to *NAT* (Network Address Translation).
- Usually, you assign machines on your LAN a private IP address, the avalible subnets are:

| Subnet | IP Range | Addresses |
|---|---|---|
| 10.0.0.0/8 | 10.0.0.0 – 10.255.255.255 | 1677216 |
| 172.16.0.0/12 | 172.16.0.0 – 172.31.255.255 | 1048576 |
| 192.168.0.0/16 | 192.168.0.0 – 192.168.255.255 | 65536 |

  Most consumer routers use a 24-bit subnet in the 192.168.0.0/16 block, so it is usually safe to use one within your ISP's network. It is not unusual that an ISP will use a subnet in 10.0.0.0/8. The 172.16.0.0/12 range is fairly rare to come by, so a subnet in there should be safe as well. There are no rules governing the use of private subnets, so your ISP might use subnets from all of them. You might have to experiment to find an unused subnet.
- If you have to pick a private subnet, I would recommend one of the early 24-bit subnets in 192.168.0.0/16, like 192.168.1.0/24 or 192.168.2.0/24.

## Subnet Usage

- If you plan to have both a wireless and a wired network, you can have one 24-bit subnet for each, or you can divide up a 24-bit subnet into two 25-bit subnets. (There are more options, but these are the most straight forward)
- The two 25-bit subnets from 192.168.1.0/24 would be as follows:

| Usable IP's | Broadcast Address | Netmask |
|---|---|---|
| 192.168.1.1 – 192.168.1.126 | 192.168.1.127 | 255.255.255.128 |
| 192.168.1.129 – 192.168.1.254 | 192.168.1.255 | 255.255.255.128 |

- This allows us to use fewer addresses (reducing possibilities of collisions with our ISP), but still have the technical advantages of having two subnets.

## Kernel Routing Options

- The kernel has a large array of built-in options that affect a router, from blocking some forms of attacks to simply allowing us to act as a router.
- Unfortunately there are far more options than time, so I will simply mention one.
- `/proc/sys/net/ipv4/ip_forward`
- By echoing '1' into this file we enable forwarding, echoing '0' will disable forwarding.
- To act as a router, *forwarding must be enabled*.

## Configuring Your Network Devices

- Now, lets suppose we have a public, static address, provided by dhcp, of 42.42.42.42, and we've divided into 2 25-bit subnets as described above. We have eth0 as the external NIC, eth1 as the internal wired, and eth2 as the wireless.
- Configure each as follows:

```
# ifup eth0
# ifconfig eth1 192.168.1.1 netmask 255.255.255.128 broadcast 192.168.1.127 up
# ifconfig eth2 192.168.1.129 netmask 255.255.255.128 broadcast 192.168.1.255 up
# iwconfig eth2 mode master essid mynetwork
```

- Now the command `route -n` should only have one gateway, which is provided by your ISP.
- If your distro doesn't have `ifup`, use the equivalent for eth0, just be sure to be getting you're address from dhcp.

## Iptables: Introduction

- Iptables is the current method of implementing a Linux firewall. It succeeds ipchains, and is avalible in the 2.4 and default in the 2.6 kernel series.
- Firewall rules are inserted into the kernel in various tables, setup as *chains* of rules.
- With iptables you specify the table you are adding the rule to, and the chain you wish to put the rule in. There are a number of default tables, but we will only worry about two of them:
  - filter – The default table, containing the chains INPUT, OUTPUT, and FORWARD.
  - nat – Used to alter packets before or after the filter table, contains the chains PREROUTING, OUTPUT, and POSTROUTING.

## Iptables: Introduction

- In each table there are *chains* of rules. These chains execute rules in order.
- Each chain has a *default policy* for determining what to do with a packet it encounters. The policy can be any of the possible TARGETs. Some useful ones are:
  - ACCEPT – Let the packet through.
  - DROP – Drop the packet, no notification, just toss it aside.
  - REJECT – Reject the packet, with an error message.
  - REDIRECT – Send the packet else where.
- Each rule has a number of parameters it tries to match. If the packet is a match the rule applies to that packet.
- To allow all SSH traffic in to the firewall machine you would:

```
# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

## Understanding Iptables Commands

- The following rule will be explained:

```
# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

- `-A INPUT` – This says to append this rule to the chain INPUT in the default table.
- `-p tcp` – Specifies the protocol to be tcp. udp packets will be ignored by this rule.
- `--dport 22` – If we've picked a protocol, we can specify a destination port, or a source port with `--sport`. So in this case, if the packet is a tcp packet going to port 22 it is a match.
- `-j ACCEPT` – *Jump* to the target *ACCEPT* if we match.

## Using the Right Chain

- Often the difficulty with iptables is knowing when to use which chain when filtering. Here's some simple rules of thumb.
  1. If the packet is coming directly to the machine running the firewall, INPUT.
  2. If the packet is being generated by the machine running the firewall, OUTPUT.
  3. If the packet is simply passing through from one machine onto another , FORWARD.
  4. If the packet needs to been seen as a coming from a different IP address, use the *nat* table with POSTROUTING.
  5. If the packet needs to go to a different address, use the *nat* table with PREROUTING.

## Using the Right Chain, cont.

- Examples of each:
  1. Stop http access to the firewall:
     ```
     # iptables -A INPUT -p tcp --dport 80 -j DROP
     ```
  2. Don't let telnet out of the firewall:
     ```
     # iptables -A OUTPUT -p tcp --dport 23 -j DROP
     ```
  3. Allow ssh to the internal LAN:
     ```
     # iptables -A FORWARD -p tcp --dport 22 -j ACCEPT
     ```
  4. All internal machines are private, NAT to firewall's public:
     ```
     # iptables -t nat -A POSTROUTING -i eth1 -j SNAT \
      --to-source 42.42.42.42
     ```
  5. HTTP needs to forward on to a NAT'ed box:
     ```
     # iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 \
      -j DNAT --to-destination 192.168.1.15
     ```

## Basic Routing Commands

- If we are forwarding packets from other machines on to the internet, we need to know a few basic commands for iptables. We will assume that you have private addresses on your LAN, and only one public address that your firewall has.
  - eth0 is the external network device, eth1 is the internal device
  - Our public address is 42.42.42.42
  - Our box is at 192.168.1.15
- We want the world to know where our LAN's packets are coming from, so we will add and entry to the nat table:

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 42.42.42.42
```

- We want to forward SSH packets to our box:

```
# iptables -t nat -A PREROUTING -i eth0 -d 42.42.42.42 -p tcp --dport 22 \
 -j DNAT --to-destination 192.168.1.15
```

## Basic Firewall Rules

- A firewall can run in two basic modes, accept or drop.
  - In accept, we set the INPUT, FORWARD, and OUTPUT chains with a default policy of ACCEPT.
  - In drop, we set the INPUT, FORWARD, and OUTPUT chains with a default policy of DROP. Be sure to only set the policy after the rules are in place.
- If we set the policy to drop, we must explicitly allow what traffic we want. This is the most secure option, but takes more work. The following rules need to exist in a drop policy environment:

```
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A OUTPUT -o lo -j ACCEPT
# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

## Basic Firewall Rules Cont.

- If we are forwarding SSH to our box, we need to allow SSH to be forwarded:

```
# iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT
```

- And finally we need to add a rule to allow outbound traffic:

```
# iptables -A FORWARD -o eth0 -j ACCEPT
```

- In an accept policy environment, we only need to include explicit drop commands and DNAT/SNAT rules.

## Some Tips for Using Iptables

- Iptables isn't always the easiest thing to use, and you probably want to have things start automatically. So there are somethings you should do:
  1. Make an init script for starting, stopping, and restarting the firewall at boot, and whenever else.
  2. Put your rules into a script to save them. It could be in the init script itself, a sourced shell script, or you could execute another script in your favorite scripting language.
  3. Remeber to reduce typing by making functions of common commands, put executables and other things as variables, include error checking, and make nice status output.
  4. In your init script you should set your various /proc settings and load the iptables modules you will need.
  5. Always flush out your tables and set the default policy correctly when starting/stopping.

## The Most Helpful Iptables Commands

- This is a great boon to firewall creators:

```
# iptables <some settings until you're not sure what you can do> -h
```

- For example:

```
# iptables -p tcp -h
# iptables -m state -h
# iptables -j REJECT -h
# iptables -m recent -h
```

- Be sure to read the man page, and look at the example scripts that should be included with this presentation.
- Also you can use `iptables -L` to view the rules in the tables. (`iptables -t nat -L`)

## An Example Script

```
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables --flush
iptables -t nat --flush
iptables --delete-chain
iptables -t nat --delete-chain

iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 42.42.42.42
iptables -t nat -A PREROUTING -i eth0 -d 42.42.42.42 -p tcp --dport 22 \
 -j DNAT --to-destination 192.168.1.15

iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -A OUTPUT -j ACCEPT
iptables -A INPUT -i eth1 -j ACCEPT

iptables -A FORWARD -o eth0 -j ACCEPT
iptables -A FORWARD -d 192.168.1.15 -p tcp --dport 22 -j ACCEPT

iptables -P INPUT DROP; iptables -P FORWARD DROP; iptables -P OUTPUT DROP;

echo 1 > /proc/sys/net/ipv4/ip_forward
```

## An Example Script Explained

- This script will:
  1. SNAT all internal machines to our public address.
  2. DNAT and forward SSH to our box.
  3. Allow the firewall to send out packets.
  4. Allow the firewall to receive new connections on its internal interface.
  5. Allow outbound traffic from within the LAN
  6. Allow established traffic to continue.

## A Comment on NATing

- If you NAT your machines, the SNAT target works great if you have a static IP. Most broadband ISPs offer static IPs. Even when they won't offer one, you usually will have a relatively static IP. Comcast has been a change about every 2 months.
- To help keep your IP the same, be sure you DHCP client stays running so you keep requesting the same address.
- But if your IP changes, you have to change your SNAT statement in your scripts.
- iptables provides the MASQUERADE target for dynamic IPs, simply replace your SNAT line with:

```
iptables -t nat -A POSTROTING -o eth0 -j MASQUERADE
```

## Wireless: Getting A Card

- To act as your own access point, you need a network card that can be switched into *master* mode under Linux.
- The following cards are currently (December 10, 2006) supported:
  1. Prism 2/2.5/3 with the HostAP driver.
  2. Prism54 based cards
  3. Atheros cards using the madwifi driver
- I recommend the ALLNET ALL0271 802.11g PCI Card, available at http://www.allnet-usa.com/html/shop.php?kat=WiFi+54Mbit. It uses the prism54 driver.
- Please note that many cards that use the prism54 chipset are the "softmac" version, and will not work with the prism54 driver.

## Security Options

- WEP – everything supports it, but it is *very* easily cracked.
- MAC Restriction – Only approved MAC addresses can join, but you can fake an approved MAC address.
- WPA – The new security method, requires more for setup, is much more secure, but not all cards support it.
- OpenVPN – Require all clients to be part of an OpenVPN network, requires more effort on the user's part, but is supported regardless of card or OS and is very secure.

## Using WEP

- WEP is simple to setup, all you have to do is set it with iwconfig, either as hex or a string.

```
# iwconfig eth2 key restricted s:thisisacoolpassphrase
# iwconfig eth2 key restricted dead-beef-fed2-dad1
```

## Using MAC Restriction

- MAC Restriction is more difficult, but can be done. There are two methods.
- First, you can have your dhcp server offer only to listed MAC's. But then you can just set a static IP in the right range, and it will work.
- Second, you can use the mac match extension in iptables against a list of macs, and drop if it is not in the list.
- Both of these require you maintaining a list of approved macs, and can be gotten around, if you know what you're doing. Luckily, most people don't.

## MAC Restriction Example

- To use MAC restriction, we need to add iptables rules to our firewall that only allow out approved addresses.
- Remove from our previous examples the following line:

```
iptables -A FORWARD -o eth0 -j ACCEPT
```

- Now for every machine you want to allow out, add:

```
iptables -A FORWARD -o eth0 -m mac --mac-source XX:XX:XX:XX:XX:XX -j ACCEPT
```

Replacing XX:XX:XX:XX:XX:XX with the MAC address.

## Using WPA

- WPA is the “right” way to secure a WAP. Unfortunately it is no cake walk to setup, and not all drivers support it.
- You will need to install hostapd to offer WPA, and your Linux clients will need wpa_supplicant.
- hostapd is a bit of voodoo to setup, and is still fairly new. This makes it hard to debug errors you receive.
- For the time being, WPA isn’t a good solution.

## OpenVPN

- OpenVPN is a much more complicated setup, but is very powerful.
- Each client will have to have an OpenVPN client installed, and have a configuration file and key provided to it.
- This makes it difficult to add new people quickly, but is the most secure and seamless method. Unfortunately it is fairly involved, and we don't have time to cover it.
- Generally consider OpenVPN if you don't want to worry about getting WPA working, and want more security than MAC restriction.

## Feature Providing Services

- Now, putting in IP addresses manually on each client is lame. So we're going to need a dhcp server.
- Also, wouldn't it be cool to be able to reference our machines by name, not by IP?
- Heck, we might as well start filtering web traffic too.

## Installing a DHCP Server

- I recommend dhcpd version 3 for a dhcp server.
  - dhcp3-server in Debian
  - dhcp in Gentoo
  - dhcp in Fedora/Redhat
- After installing you'll need to edit dhcpd.conf.
- You'll want to set two general configuration values:

```
default-lease-time 18000 # 5 hours
max-lease-time 18000 # 5 hours.
```

## Configuring DHCP Subnets

- We need to define a subnet, using our examples thus far, we'll set up one for 192.168.1.0/25, we will act as a caching nameserver as well.

```
subnet 192.168.1.0 netmask 255.255.255.128 {
    range 192.168.1.2 192.168.1.126;
    option domain-name-servers 192.168.1.1;
    option routers 192.168.1.1;
    option broadcast-address 192.168.1.127;
    option subnet-mask 255.255.255.128;
}
```

- Now for the other subnet:

```
subnet 192.168.1.128 netmask 255.255.255.128 {
    range 192.168.1.130 192.168.1.254;
    option domain-name-servers 192.168.1.129;
    option routers 192.168.1.129;
    option broadcast-address 192.168.1.255;
    option subnet-mask 255.255.255.128;
}
```

## Static Addresses with DHCP

- Sometimes we want a machine to have a static address, but still want to use dhcp. First we have to remove the address from the `range` statement in the proper subnet. Usually, you should leave the easy, low number addresses for static machines.
- So we'll adjust the first subnet's range to be 192.168.1.25 192.168.1.126, and then add the following:

```
host mybox {
    hardware ethernet 00:0F:0D:56:12:FF;
    fixed-address 192.168.1.10;
}
```

## Running A Caching Nameserver

- To start with, we'll setup a nameserver that simply acts as a relay for our network. It only caches lookups to save time, and makes our job easier for DNS within the LAN.
- We need to install bind version 9, usually in a package called 'bind9'.
- Some systems include more default configuration than others. In Debian we only need to add a few lines to the correct configuration file. Usually `named.conf` or `named.conf.local` or `named.conf.options`.

```
acl mynets { 192.168.1.0/24; };
options {
    allow-query { mynets; localhost; };
};
```

- This allows our LAN to query the nameserver and receive responses.

## Providing Names to Local Machines

- If you're like me, you get tired of having to type in IP addresses to access machines. There's two ways to tie names to IPs:
    1. Using `/etc/hosts`.
    2. DNS
- We can set up DNS to provide hostnames by either using a domain we own, or use an unused, "safe" one.
- Each machine in our LAN will be then be machinename.example.com.
- We will provide a default search path to allow machines to be accessed simply by machine name.

## Configuring Your Zone

- If you already run your own nameserver, simply provide names for IP addresses in your private network, and add a reverse lookup for the internal IPs on your firewall.
- If you don't have a domain, we can use mystuff.example.com, since it will never exist. Our machines will be machine.mystuff.example.com.
- Add in your dhcpd.conf the line:

  ```
  option domain-name "mystuff.example.com"
  ```

  This will set the default search for your clients.

## Configuring Your Zone

- Something like this for a zone file should do the trick:

```
@       IN      SOA mystuff.example.com (
                1
                3h
                1h
                1w
                1h )

@       IN      NS      mystuff.example.com.
        IN      A       192.168.1.1

mybox   IN      A       192.168.1.15
another IN      A       192.168.1.25
```

- Now you'll need to add the zone to your named.conf:

```
zone "mystuff.example.com" {
    type master;
    file "/whereever/bind/is/mystuff.example.com";
};
```

## Bind Notes

- Please note, I've glossed over some details if you running your own bind server.
- You need a file to reference other servers to pull down DNS information from, most distros probably come with this in the default configuration files.
- You will need to have a reverse lookup file for 127.0.0, once again this is often included.
- Finally, you will need to make a reverse lookup file for your LAN if you choose to use DNS to give names to your machines.
- If all works correctly a `host mybox` will resolve to mybox.mystuff.example.com which will be 192.168.1.15.

## Filtering Web Traffic

- If we want to filter our web traffic to stop porn, amongst other things, we can.
- SquidGuard (http://www.squidguard.org/), and DansGuardian (http://dansguardian.org/) provide this ability.
- Both of them can be used to block URLs based on blacklists.
- DansGuardian can filter based on the content of the page, matching by regular expressions among other things.

## Installing

- First we need to install Squid. Luckily Squid is available in most distros.
- For SquidGuard, add the following to the `squid.conf`:

```
redirect_program /prefix/bin/squidGuard
redirect_children 4
```

## DansGuardian Configuration

- In the DansGuardian conf file, try these:

```
filterip =
filterport = 8080
proxyip = 127.0.0.1
proxyport = 3128 # default squid port
```

- Once you have it setup, and if you're listening on port 8080, add this rule before any SNAT rules for the PREROUTING chain:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT \
 --to-port 8080
```

## Notes on Hardware

- Remember, just having two NICs and a wireless card won't solve all your problems. If you want a lot of machines to hook up to your wired network, you'll need a switch.
- Sometimes you can get a all-in-one router to act as a switch, but I've never had much luck with that.
- 5-port 10/100 switches run about $25 to $40 depending on where you get them. 16-port 10/100 switches run about $40 to $60.
- 5-port 10/100/1000 switches run about $40 to $60. 16–24-port 10/100/1000 switches run about $80 and up.

## Notes on Machines

- If you don't have an old machine lying around, and want something a little cheaper you can look into using a Linksys router.
- A number of the Linksys routers can have custom firmware loaded so they're running Linux. From there you can do everything we've described here.
- OpenWRT (http://openwrt.org) is the project that you'll want to look into for what routers are now supported. You have to be careful, not all versions of the same model may work.
- More information can be found on the OpenWRT site.

## Resources

- General References:
  1. Configuring a Linux Wireless Router (http://martybugs.net/wireless/router.cgi).
  2. Firewall and Proxy Server HOWTO (http://www.tldp.org/HOWTO/Firewall-HOWTO-6.html).
- Iptables:
  1. Using Linux iptables or ipchains to set up an internet gateway (http://yolinux.com/TUTORIALS/LinuxTutorialIptablesNetworkGateway.html).
  2. Iptables Tutorial (http://iptables-tutorial.frozentux.net/iptables-tutorial.html),
  3. O'Reilly's *Linux Server Security*, by Michael D. Bauer.
  4. *Troubleshooting Linux Firewalls* by Michael and Scott Shinn.

## Resources

- BIND:
    1. DNS HOWTO (http://tldp.org/HOWTO/DNS-HOWTO-3.html).
    2. Bind Administrator Reference Manual (http://www.bind9.net/manual/bind/9.3.2/Bv9ARM.ch01.html).
    3. O'Reilly's *DNS and BIND* by Paul Albitz and Cricket Liu.