Bridge + Firewall + DSL Mini−HOWTO

Table of Contents

Bridge + Firewall + DSL Mini−HOWTO

Derek Ney derek@hipgraphics.com

Nov 9, 2000

Configuring a Linux system to act as a firewall and bridge with a DSL network connection

1. Introduction

1.1 History

This document was started on December 10, 1999 by Derek Ney derek@hipgraphics.com after three day's worth of frustration with bridging and firewalling after switching from a PPP network link to a DSL link.

1.2 New versions

The newest version may be found in different formats at the LDP homepage http://www.linuxdoc.org/.

Version History

v0.04 (Nov 9, 2000)

- Updated for newer bridge configuration utility bridgex

v0.03 (Mar 24, 2000)

- Fixed up URL for BRCFG.tgz

v0.02 (Dec 13, 1999)

- Incorporate revisions from Leonard Dickens (thanks Leonard!)

v0.01 (Dec 10, 1999)

- Initial version

1.3 Copyrights

2. Bridging, Firewalls, and DSL connections

Until recently, our local network was hooked into the global net via PPP over a modem. I had installed a firewall using IPChains (http://www.linuxdoc.org/HOWTO/IPCHAINS−HOWTO.html) with this setup and it worked nicely. We recently upgraded to a DSL connection. I thought it would be trivial to simply switch my firewall to insulate me from the larger net coming in via the DSL connection. I was wrong. It took three days of work to finally get it up and running. I found a lot of suspect information on the net that caused a good deal of confusion. This mini−HOWTO was written because I suspected that our setup will be a quite common configuration in the future when DSL becomes more widespread and I wanted to help people avoid massive frustration.

I guess this is applicable to a cable modem setup, but YMMV as I know nothing about cable modem hookups.

2.1 The Problem

The problem I am trying to solve is to configure the system such that the firewall code in the kernel (that is manipulated with ipchains) can be used to filter the packets that travel back and forth between the outside world and the local network. I also needed some of the local machines to be "seen" on the global net (though always filtered through the firewall). This ruled out IP masquerading (see IP Masquerade HOWTO) which would otherwise probably be a simpler solution. This is not as simple as it seems.

2.2 The Solution

To accomplish our goal of insulating a local net from the global net (over DSL) by using our Linux box, we will use two ethernet (NIC) cards. One card is hooked up to the local net and one to the global net. The only machine that can directly talk to the outside world is the Linux box. All other machines in our local net must go through the Linux box (firewall).

Configuring the software really consists of two problems:

- Route packets between the local and global net (bridging)
- Filter the packets to stop some from traversing the firewall

The Bridging mini−Howto gives detailed instructions that solves the first problem by routing packets between the two sides of the network (local and global). This works by putting both NIC's into "promiscuous" mode such that they sniff all the packets on each NIC and transfer packets over when they belong on the other side. This is done transparently; the other computers on the net do not even see the bridge, because it does not even have an IP address. But this does not totally solve the problem. I wanted the firewall to have an IP address (for administration via the network, if nothing else) and more importantly, the bridge code in the kernel intercepts and bridges packets BEFORE they get to the firewall code, so the firewall will have no effect.

It turns out you can assign your NIC's IP addresses and still use them as a bridge. Although the Bridging mini−Howto does not do this (well actually, it uses loopback addresses), it works fine. That solves one problem. For the firewall problem, we turn to a fine kernel patch at http://ac2i.tzo.com/bridge_filter/ that causes the firewall rules to be invoked for packets that are being bridged with a special new rule "bridgein".

2.3 Setup Overview

This mini−HOWTO is meant to handle the situation where you have a Linux box configured as a gateway/firewall. The system has 2 NIC cards installed. One of the NIC cards is connected to the outside world (in our case a DSL modem) and nothing else. The other NIC is connected to our local network.

Note that I have only had one experience with this and it was on my i386 (ABIT BP6 MOBO, w/2 celery) box with RedHat 6.0 with the 2.2.13 kernel, and a DSL modem going to a router, and two Netgear FA310TX NIC cards. Your mileage may vary.

Also note that the steps here will leave your network open to potential attack during setup (before the firewall is turned on). If you are very paranoid you will want to take extra steps to avoid this.

2.4 References

I found a good deal of information on the net that I used to finally get things working. Some of the information was useful, but inaccurate.

The Bridging mini−Howto was instrumental in getting things up. Unfortunately using it alone does not implement a firewall.

The Linux Bridge+Firewall mini−HOWTO at first looked like just what I needed. However, it turns out that I think it is inaccurate. I got things sort−of working with it, but in the end I realized that it was not necessary to split your sub−net in two like it directs and did not use that method. If you look at this document, take it with a grain of salt.

The Bridge Filter Patch is the key to getting the whole thing to work. Oddly enough, the information on the web page directs you to the Bridge+Firewall mini−HOWTO. You do not need to use the information in Bridge+Firewall mini−HOWTO to get things to work. You will need this patch.

The IPCHAINS HOWTO is invaluable in setting up the firewall itself. I do not attempt to cover the details of firewall setup in this document; only issues which are different because of the bridging setup are mentioned here.

3. Procedure

The basic procedure is as follows:

- Setup your hardware (and verify that it works)
- Patch and configure the kernel
- Configure your network (ifconfig, route, bridging)
- Configure the firewall

3.1 Example Setup

Throughout this procedure, I will assume a setup with two ethernet (NIC) cards, an outside link via DSL (where a DSL modem connects to one of the NIC's), and a local net that connects to the other NIC. I will arbitrarily call the NIC to the DSL modem "eth1" and the local net NIC "eth0". The device naming by the kernel of the NIC's depends on what slot they are in.

I will assume that you have been assigned a subnet of IP addresses at 192.168.2.128−191, i.e. a netmask of 255.255.255.192, and the router provided by the DSL company is at 192.168.2.129. These are all arbitrary fictional examples to illustrate the setup. I will use the address 192.168.2.130 for the firewall machine (both NIC's), though it turns out you can also use distinct IP addresses for each NIC if you want.

3.2 Hardware Setup

You will need two ethernet cards to make this work. The biggest problem I had was that I randomly picked a slot in my motherboard for the second NIC and it turned out that that slot (PCI) shared an interrupt with the first NIC. I did not know that this was a problem (in fact there is little information about this, and I thought it should work fine). It caused both cards to shut down quietly (no error indication) and stop sending and receiving packets. Naturally when you are doing all sort of configuration changes, this is the last thing you need. I do not know if this is a problem with all PCI NIC cards or just ours, but I would advise against sharing interrupts. The tulip driver, which we use, reports the IRQ for each NIC in syslog when you boot. There is a bunch of information out there (see the Ethernet−HOWTO section Using More than one Ethernet Card per Machine) about making the kernel recognize two ethernet cards using boot arguments; however, I did not need this (my kernel recognized both cards with no arguments).

Next, you need to hook the second NIC to the DSL modem (or whatever links you to the outside world) and make sure that it is working. You should be able to ifconfig the second ethernet card to a proper IP address and ping the router on the other end of your outside link. This verifies that you can send and receive packets over the DSL link. For instance, for the sample net you would do:

```
  ifconfig eth1 192.168.2.130 netmask 255.255.255.192 broadcast 192.168.2.191
```

to configure the NIC. And then

```
  ifconfig eth0 down # just to make sure it does not interfere with things
  ping 192.168.2.129
```

to test that you can get to the router. For good measure, you should also test that you can get to the machines on your local network through the other NIC:

```
  ifconfig eth1 down # just to make sure it does not interfere with things
  ifconfig eth0 up
  ping 192.168.2.x # where x is the address for a machine on your local net
```

At this point, you have verified that all the hardware is working.

3.3 Bridge Config

Depending upon your kernel version you will need either the old bridge configuration utility (BRCFG) for kernels before 2.2.14, or the new bridge configuration utility (bridgex) for later kernels; these utilities allow you to control the bridging in your kernel when CONFIG_BRIDGE is turned on. **BRCFG** is distributed as source with pre−compiled executables. I do not know what kernel the executable was compiled with, but I got different results after I recompiled it with my kernel (2.2.13) include files. Unfortunately, to do this I had to patch them slightly. Here are the patches:

```
diff −C 3 −r /tmp/BRCFG/brcfg.c ./brcfg.c
*** /tmp/BRCFG/brcfg.c  Wed Feb 21 19:11:59 1996
−−− ./brcfg.c   Wed Dec  8 12:52:23 1999
***************
*** 1,6 ****

! #include <sys/types.h>
! #include <sys/socket.h>
  #include <skbuff.h>

  #include "br.h"
−−− 1,6 −−−−

! #include <types.h>
! #include <socket.h>
  #include <skbuff.h>

  #include "br.h"
```

Apply the patch, recompile **brcfg** and install it somewhere appropriate (I chose **/usr/sbin**).

For kernels later than 2.2.13 you definitely want to use the newer bridge configuration utility bridgex. I am not sure if it works with earlier kernels or not. Not that the URL for this utility is found in the kernel configuration help file **/usr/src/linux/Documentation/Configure.help**, so if the URL mentioned here is not correct, look in the help file (it is the help for the **CONFIG_BRIDGE** kernel configuration item. The bridgex tarball contains an already compiled executable, but you should probably remake it using the included Makefile. Note that the bridgex utility takes slightly different arguments than does the BRCFG package (that will be covered later when I talk about configuring the bridge).

3.4 Kernel Configuration

You will need to patch and configure your kernel for bridging and the bridging filter (as well as firewalling, networking, etc. if you do not already have it). The following kernel configuration items will be needed (at least):

```
  CONFIG_EXPERIMENTAL=y
  CONFIG_BRIDGE=y
  CONFIG_FIREWALL=y
  CONFIG_IP_FIREWALL=y
```

You should grab the Bridge Filter Patch and apply it to your kernel. Recompile and install your kernel and then reboot.

3.5 Putting It All Together

So you should have your two NIC's working, a newly configured kernel, and **brcfg** installed. Now you need to construct a startup script to put it all together. I did this using the RedHat type startup scripts (**/etc/rc.d**). I put specific network addresses and masks in **/etc/sysconfig/network**:

```
 GATEWAY=192.168.2.129          # the address of the DSL router
 GATEWAYDEV=eth1                # the NIC that the router is connected to
 ETH0_ADDR=192.168.2.130        # the IP address for the NIC on our LAN
 ETH0_MASK=255.255.255.192      # the netmask of our LAN
 ETH0_BROAD=192.168.2.191       # the broadcast address of our LAN
 ETH1_ADDR=192.168.2.130        # the IP address for the NIC on the DSL side
                                # can be different from ETH0_ADDR if you want
 ETH1_MASK=$ETH0_MASK           # the DSL side netmask, should be the same as eth0
 ETH1_BROAD=$ETH1_BROAD         # ditto for the broadcast address
```

Next I created a script in **/etc/rc.d/init.d/bridge** to setup the bridge. I include two scripts here. The first script is used with the old BRCFG utility, the second for the newer bridgex. First the one for the older BRCFG:

```
#!/bin/sh
#
# bridge      This shell script takes care of installing bridging for dsl with BRCFG
#
# description: Uses brcfg to start bridging and ifconfigs eths
# processname: bridge
# config:

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
```

```
. /etc/sysconfig/network

# See how we were called.
case "$1" in
  start)
        echo −n "Configuring bridge: "
        ifconfig eth0 $ETH0_ADDR netmask $ETH0_MASK broadcast $ETH0_BROAD
        ifconfig eth1 $ETH1_ADDR netmask $ETH1_MASK broadcast $ETH1_BROAD
        route add $GATEWAY dev $GATEWAYDEV
        route add default gw $GATEWAY dev $GATEWAYDEV
        ifconfig eth0 promisc
        ifconfig eth1 promisc
        brcfg −enable
        echo
        ;;
  stop)
        # Stop daemons.
        brcfg −disable
        ifconfig eth0 down
        ifconfig eth1 down
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  status)
        ifconfig eth0
        ifconfig eth1
        brcfg
        ;;
  *)
        echo "Usage: bridge {start|stop|restart|status}"
        exit 1
esac

exit 0
```

The next script is the one to use with the newer bridge configuration utility bridgex. Note that bridgex is much more configurable than the older BRCFG and so you may want to look man page included with the bridgex tarball and custom configure this script:

```
#!/bin/sh
#
# bridge      This shell script takes care of installing bridging for dsl with BRCFG
#!/bin/sh
#
# bridge      This shell script takes care of installing bridging for dsl with bridgex
#
# description: Uses brcfg to start bridging and ifconfigs eths
# processname: bridge
# config:

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# See how we were called.
case "$1" in
```

```
  start)
        echo −n "Configuring bridge: "
        ifconfig eth0 $ETH0_ADDR netmask $ETH0_MASK broadcast $ETH0_BROAD
        ifconfig eth1 $ETH1_ADDR netmask $ETH1_MASK broadcast $ETH1_BROAD
        route add default gw $GATEWAY dev $GATEWAYDEV
        ifconfig eth0 promisc
        ifconfig eth1 promisc
        brcfg start
        brcfg device eth0 enable
        brcfg device eth1 enable
        echo
        ;;
  stop)
        # Stop daemons.
        brcfg stop
        ifconfig eth0 down
        ifconfig eth1 down
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  status)
        ifconfig eth0
        ifconfig eth1
        brcfg
        ;;
  *)
        echo "Usage: bridge {start|stop|restart|status}"
        exit 1
esac

exit 0
```

The script is run during bootup. It assigns addresses to each NIC, adds a default route that goes to the DSL router, adds a specific route direct to the DSL router, puts each NIC in "promiscuous" mode, and then enables bridging. I linked this script into the following directories in **/etc/rc.d**:

```
 /etc/rc.d/rc0.d/K90bridge
 /etc/rc.d/rc1.d/K90bridge
 /etc/rc.d/rc2.d/S11bridge
 /etc/rc.d/rc3.d/S11bridge
 /etc/rc.d/rc4.d/S11bridge
 /etc/rc.d/rc5.d/S11bridge
 /etc/rc.d/rc6.d/K90bridge
```

This makes it run right after the network start script. You should disable other configuration of eth0 (or eth1) such as done in the **/etc/rc.d/init.d/network** script (in RedHat by removing files **ifcfg−eth?** from **/etc/sysconfig/network−scripts/**).

To try things out, I suggest rebooting in single user mode (specify **"single"** as an arg to the kernel, e.g. in lilo "lilo: linux single") and running the startup scripts in **/etc/rc.d/rc3.d** one at a time until you get to the bridge startup. Startup the bridge and then see if you can reach some machines (you probably want to use "**ping −n**" for this to keep the nameserver out of the equation):

- ping the DSL router

- ping a local machine
- ping a machine on the global net

If you can ping all those places, there is a good chance that things are working. Note that the bridge takes a few moments to startup. You can monitor the status of the bridge by issuing the command **brcfg** with no arguments.

3.6 Firewall Setup

You still need to setup your firewall (assuming you want one) to prevent unauthorized access. The Bridge Filter Patch that you applied allows you to use a new built−in rule "bridgein" with ipchains. This rule is used whenever a packet is going to be forwarded either from eth0 to eth1 or vice versa. The bridgein rule is not used when a packet is destined for the firewall itself; you will want to use the input rule for that. I will not attempt to delve into the firewall setup in detail; please see the IPCHAINS HOWTO for that.

3.7 Local Machine Setup

For each of your local machines, you simply have to setup the proper IP address and netmask and use the DSL router for the gateway (default route). The firewall/bridge will bridge the packets to/from the DSL router.

---

4. Quirks and Problems

4.1 Odd message when using ipchains −X

The patch to add the bridgein built−in rule to ipchains makes the "delete all chains" command, **ipchains −X**, issue the following error:

```
ipchains: Device or resource busy
```

As far as I can tell this is harmless. I suspect that ipchains does not understand that the new bridgein rule is a builtin.

4.2 Shared Interrupts

As I mentioned in Hardware Setup, at least for PCI NIC's you do not want to share interrupts between the two cards (or probably with any other device).
