

Firewalling for Free:

An Enterprise Firewall Without the Enterprise Price

Name: Shawn Grimes
Date: November 25, 2001
Course: CT-401

Table of Contents

Introduction.....	1
Nature of Bridging Firewalls.....	1
Physical Setup of Bridge.....	2
Hardware.....	3
Software.....	3
Compiling and Configuring Kernel.....	4
Compiling and Configuring Bridge.....	7
Compiling and Configuring Iptables.....	8
Price Comparison.....	9
Redundancy.....	10
Attachment 1.....	11
Attachment 2.....	13
References.....	16

Businesses and organizations have become increasingly aware of the needs for adequate network security. One of the first steps in securing a network is installing a firewall to block harmful traffic. However, many managers are very hesitant to invest thousands of dollars into more IT equipment, not too mention the necessary training for IT staff to manage the new equipment. Managers can look to open source software for an effective enterprise firewall solution without the enterprise cost.

The Netfilter Project is a packet-filtering module for the Linux kernel. This addition to the kernel provides stateful packet filtering as well as packet mangling for Network Address Translation (NAT). These are two features that are commonly found in expensive closed source solutions.

Stateful packet filtering allows you to block all incoming connections that have not been established from the inside. This is useful when you wish to deny all connections to ports on the internal net. If you just deny all incoming connections to ports, your computers on the internal net will not be able to do any external browsing. A TCP/IP connection is a three-way handshake. Let's take a look at a typical Internet connection to explain the need for stateful packet filtering. Host A is on the internal network and it wants to view a web page on the web server of Host B on the Internet. A web server usually runs on port 80 of a server. Host A will send a packet directed at port 80 of Host B. The packet also has a source port included. This source port is usually greater than 1024. Ports above 1024 are commonly referred to as ephemeral ports. Most services run on a server do so on ports below 1024. This is not a requirement, just common practice to help separate the two. Host A's initial packet is sent with the SYN flag set to signal Host B it wants to make a connection. Host B will then send a packet back with the SYN and ACK flags set to let Host A that it exists and is accepting its

connection. This packets source port is 80 and the destination port is whatever Host A's initial packet's source port was. The problem exists when the packet reaches the firewall. If the firewall does not have stateful filtering, it will drop the packet. If it does have stateful filtering, it will check its que (where it stores existing connections) for an initial packet from Host A requesting the connection and then pass the packet.

A bridging firewall is one that is invisible on a network. It exists only physically on the network but unlike routers and gateways, it doesn't exist on TCP/IP (See Figure 1). With bridging, all packets from an internal network are sent down the line and through the bridge. The bridge inspects the packet and then decides to forward it on to its destination or drop it. No changes are made to the packet and as far as the hosts on either side are concerned, they are sending their packets directly to the host on the other end (See Figure 2). Until recently, a bridging firewall was unavailable for Linux systems. But thanks to Lennert Buytenhek, bridging code for Linux now exist. This software is continually being developed and updated but is only in the beta stages of development. However, my tests have found that not only is Lennert's code functional, it is suitable for production environments.

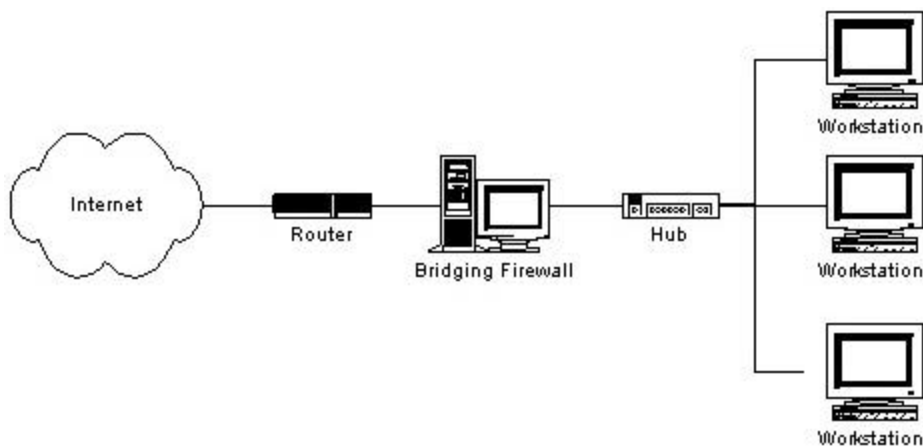


Figure 1. Bridging firewall physical location.

```
C:\WINNT\System32\cmd.exe
(C) Copyright 1985-2000 Microsoft Corp.
C:\Documents and Settings\grinessh\Desktop>tracert www.google.com
Tracing route to www.google.com [216.239.37.100]
over a maximum of 30 hops:
  1  10 ms  10 ms  10 ms  10.99.67.1
  2  10 ms  10 ms  10 ms  bb1-ge5-0.rdc1.md.home.net [24.18.95.1]
  3  10 ms  10 ms  10 ms  c1-pos4-0.bltmdd1.home.net [24.7.74.97]
  4  10 ms  10 ms  11 ms  c2-pos1-0.washdc1.home.net [24.7.65.89]
  5  10 ms  10 ms  10 ms  c1-pos10-0.washdc1.home.net [24.7.78.177]
  6  10 ms  10 ms  10 ms  ibr02-p1-0.stng01.exodus.net [209.1.169.205]
  7  10 ms  20 ms  20 ms  bbr01-g3-0.stng01.exodus.net [216.33.96.147]
  8  10 ms  10 ms  10 ms  hbr01-p4-0.stng02.exodus.net [209.1.169.198]
  9  10 ms  10 ms  10 ms  dcr02-g2-0.stng02.exodus.net [216.109.66.2]
 10  50 ms  10 ms  20 ms  csr11-ve241.stng02.exodus.net [216.109.66.90]
 11  10 ms  10 ms  10 ms  216.239.47.66
 12  10 ms  10 ms  20 ms  abovenet-van11.google.com [64.124.113.173]
 13  10 ms  10 ms  10 ms  vabil-gige-1-1.google.com [216.239.47.26]
 14  10 ms  10 ms  20 ms  www.google.com [216.239.37.100]
Trace complete.
C:\Documents and Settings\grinessh\Desktop>
```

Figure 2. Bridge is invisible in a tracert.

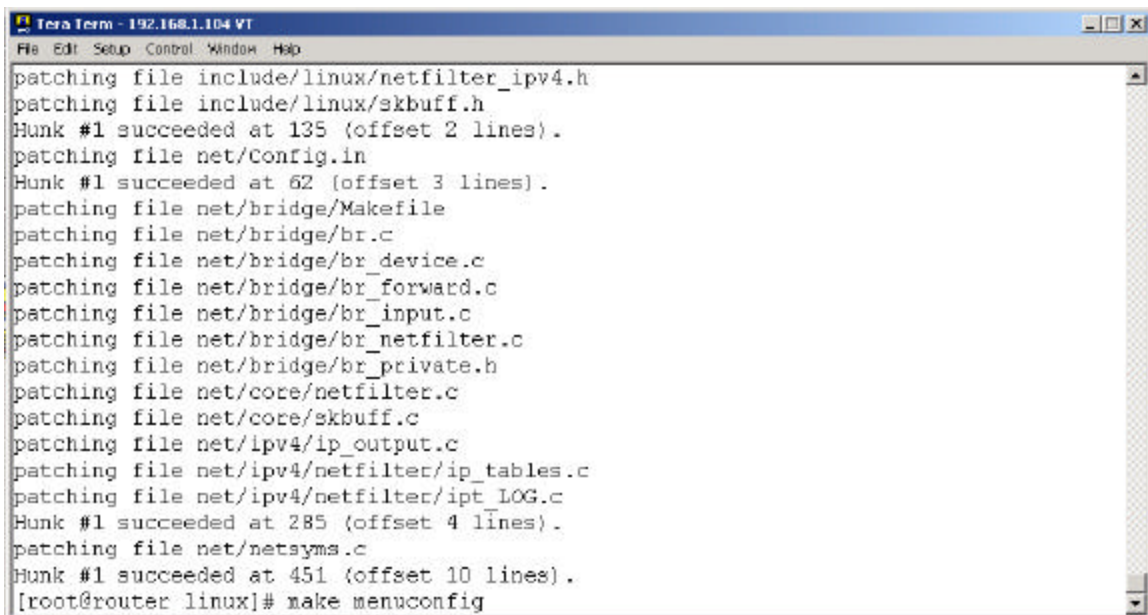
For my tests, I used a Compaq Proliant DL360 server with dual 1.3 GHz processors and 1 GB of RAM. You also need two Network Interface Cards (NIC). One going to the Internet and the other going to your internal network. On average I saw about 70Mbits of traffic on our network with no packet loss using Lennert's bridging code.

To set up the bridging firewall you'll first need to download and install Red Hat Linux 7.2 onto the host you wish to use as your firewall. Then update and install all components on the host. This will make sure you have all the up to date modules for your server and that any previous vulnerabilities in software have been patched. You'll want to secure this box as much as possible. This means not installing any modules you don't need and not enabling any services you don't want to run. There should be no other services running on the firewall. Refer to the AusCERT's guide on securing Unix found in the references section for more information on locking down Unix.

Then you will need to download the bridge utilities and bridge-kernel patch from Lennert's website, <http://bridge.sourceforge.net>. You'll also need the latest version of

Netfilter available from <http://netfilter.samba.org>. And finally you'll need the latest version of the Linux kernel from <ftp.kernel.org>, currently 2.4.14.

Copy the downloaded kernel to the `/usr/src` directory and execute `tar -xzvf linux-2.4.14.tar.gz`. This will extract the kernel into `/usr/src/linux`. Copy the downloaded Bridge-Netfilter kernel patch to this directory and apply it by executing `patch -p1 < bridge-nf-0.0.3-against-2.4.13.-ac7.diff`. (See Figure 3).



```
Tera Term - 192.168.1.104 VT
File Edit Setup Control Window Help
patching file include/linux/netfilter_ipv4.h
patching file include/linux/skbuff.h
Hunk #1 succeeded at 135 (offset 2 lines).
patching file net/Config.in
Hunk #1 succeeded at 62 (offset 3 lines).
patching file net/bridge/Makefile
patching file net/bridge/br.c
patching file net/bridge/br_device.c
patching file net/bridge/br_forward.c
patching file net/bridge/br_input.c
patching file net/bridge/br_netfilter.c
patching file net/bridge/br_private.h
patching file net/core/netfilter.c
patching file net/core/skbuff.c
patching file net/ipv4/ip_output.c
patching file net/ipv4/netfilter/ip_tables.c
patching file net/ipv4/netfilter/ipt_LOG.c
Hunk #1 succeeded at 285 (offset 4 lines).
patching file net/netsyms.c
Hunk #1 succeeded at 451 (offset 10 lines).
[root@router linux]# make menuconfig
```

Figure 3. Apply bridge patch to kernel

Within the `/usr/src/linux` directory run `make menuconfig` to begin configuring the new kernel. You will now be able to choose which options will be compiled into your new kernel. As a security precaution, I only recommend installing necessary components. The required options for a bridging firewall are found in the “Network Options” section. You’ll need to set the “Network Packet filtering (replaces ipchains)” component (See Figure 4). This will enable you the “IP: Netfilter Configuration -->” submenu (See Figure 5). In that menu you can choose the different features you will

want your firewall to have. I usually select all of them because I find that they are all very useful (See Figure 6). Now go back to the “Network Options” section and select “802.1d Ethernet Bridging” and then “netfilter (firewalling) support (NEW)” (See Figure 7). You can now exit from menuconfig and begin to compile your new kernel. At the command prompt run `make dep`. Then build the kernel image by executing `make bzImage`. Now copy the built kernel to the boot directory, `cp /usr/src/linux/arch/i386/bzImage /boot/bzImage.bridge`. The boot manager needs to be configured to recognize the new kernel image. If you use LILO this can be done by editing the `/etc/lilo.conf` file and adding a stanza similar to:

```
image=/boot/bzImage.bridge
    label=bridge
    root=/dev/hda3
    append="hdc=ide-scsi"
    read-only
    initrd=/boot/initrd-2.4.9-13.img
```

and then exit and save. Now rerun LILO to

update it's configuration, `lilo` and then `lilo -q`.

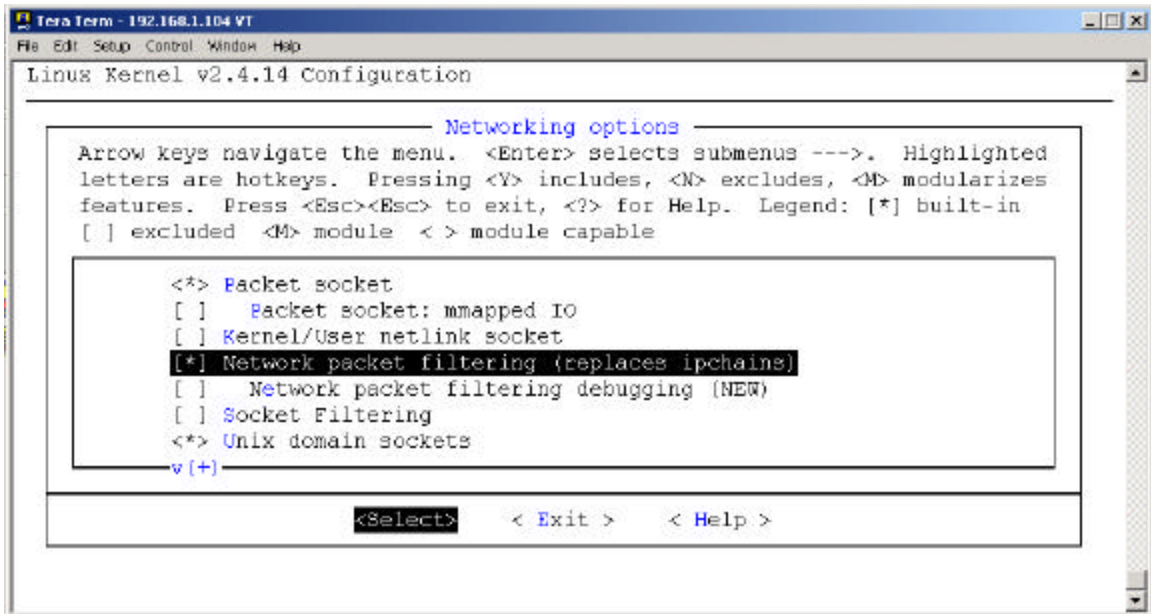


Figure 4. Install packet filtering.

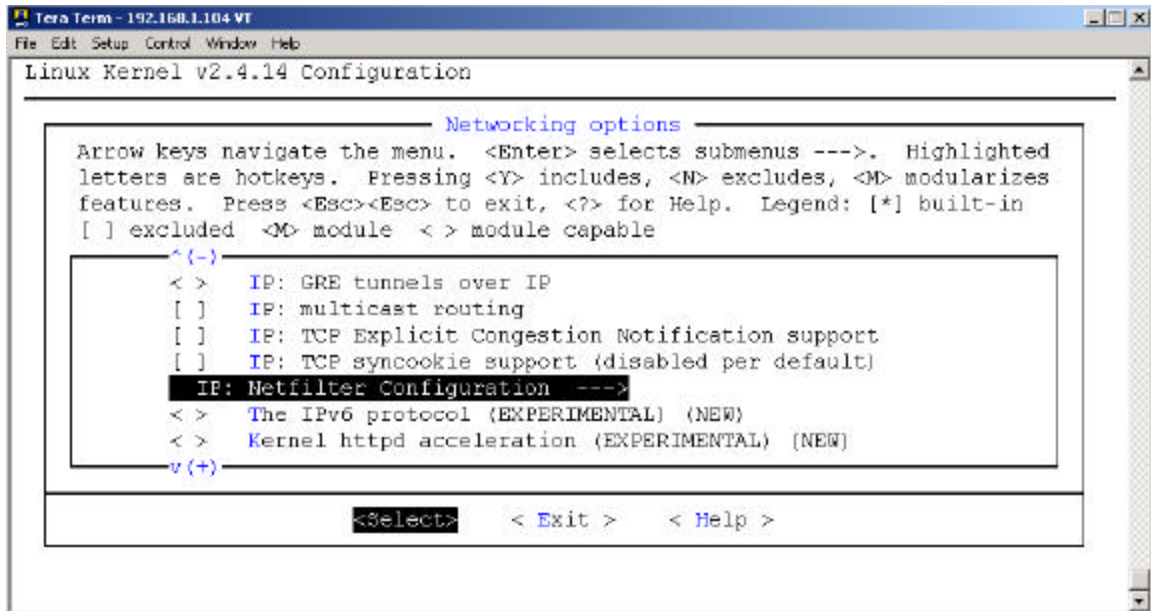


Figure 5. Netfilter Submenu

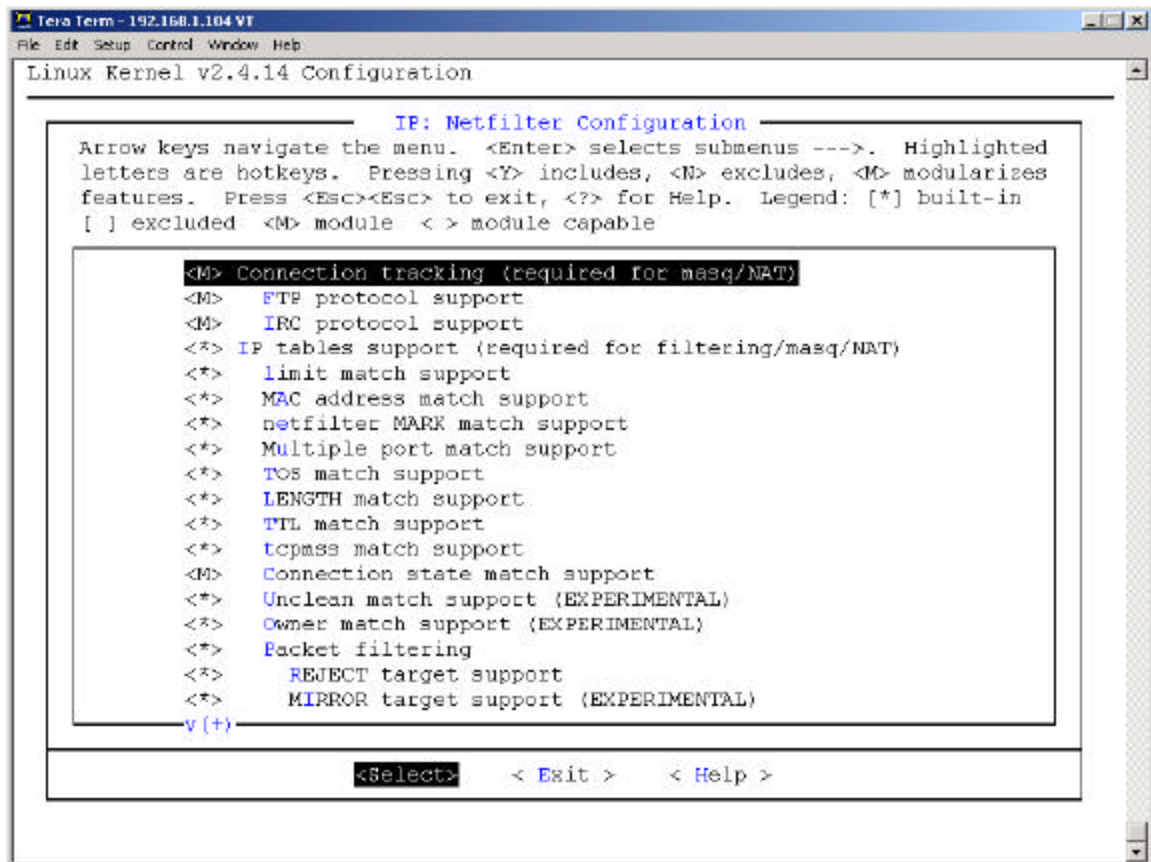


Figure 6. Netfilter Submenu Options

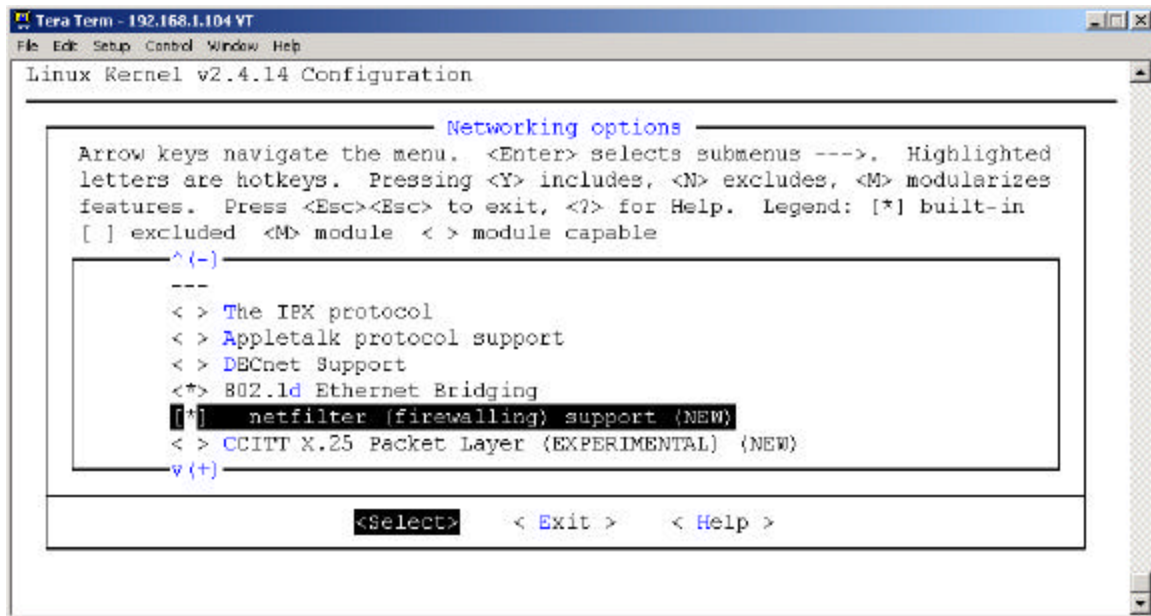


Figure 7. Bridge-firewall support

Reboot your machine and select the “Bridge” option from the boot menu. If everything is configured correctly your computer will boot and you will now have to install any modules that you chose in menuconfig. Do this by changing to the `/usr/src/linux` directory and executing `make modules; make modules_install`. If your computer does not boot, reboot the machine and choose the boot option. Rerun `make menuconfig` from the `/usr/src/linux` directory and select the right components for your system.

If the computer boots and the modules are installed, you can now start to setup the bridge. First extract the bridge utilities, `tar -xzvf bridge-utils-`

`0.9.3.tar.gz`. Compile the utilities by executing the following sequence:

```
cd ./bridge-utils-0.9.3
make
cp ./brctl/brctl /usr/bin
cp ./brctl/brctld /usr/bin
```

The utilities are now compiled and you are ready to begin building the bridge.

First, you have to create a new bridge by executing `brctl addbr mybridge`. That command will create a bridge called “mybridge.” Executing an `ifconfig` will now display “mybridge” as a network interface. You’ll have to add which NICs you want to be part of the bridge. You’ll need at least two NICs for a bridge but it’s possible to have even more to provide redundancy as explained later. To add NICs to the bridge run `brctl addif mybridge eth0` and `brctl addif mybridge eth1`. This will add NICs 0 and 1 to the bridge. The format is `brctl addif bridgename interface-name`. Typing `brctl --help` at anytime will give you a list of useful commands that you can execute to setup your bridge. Next, you need to configure your NICs to have no ip address and to operate in promiscuous mode, `ifconfig eth0 0.0.0.0 promisc` and `ifconfig eth1 0.0.0.0 promisc`. By not having an ip address assigned to the NIC you ensure that no one will be able to connect to the device via TCP/IP. This provides a very important layer of security to the firewall. The firewall rules will only be able to be configured at the local machine. This may seem like an inconvenience to the administrator but the security increase that it provides is worth the inconvenience in my mind.

You should now try to ping a host on the Internet from a host on the internal network. If your bridge is configured properly you should receive your echo replies. You can now move on to setting up and configuring the firewall portion.

First, you must extract the netfilter/iptables utility that you downloaded earlier, `tar -xzvf iptables-1.2.4.tar.bz2`. Now compile the tool by running the following commands:

```
cd ./iptables-1.2.4
```

```
make
```

```
make install
```

The first thing you can do after iptables is installed is run `iptables -L`. This will list all the current rules. The only chain that you need to concern yourself with is the Forward chain. The bridging firewall doesn't use any other chains because it is basically forwarding all traffic on. The common use of iptables is `iptables -A FORWARD pattern-to-match -j action`. The most common options for the "pattern-to-match" section are:

<code>-p, --protocol</code>	common uses are tcp,udp,icmp
<code>-s,-d</code>	source or destination address
<code>--dport,--desitnation-port</code>	destination port
<code>--sport,--source-port</code>	source port

Actions can be DROP, REJECT, ACCEPT, or LOG. I prefer to use DROP because it doesn't send anything back to a would-be attacker. Their intelligence gathering techniques are slowed drastically as they wait for their packets to time out each time instead of receiving a host unreachable error. The LOG action allows you to capture the packet headers to Syslog. There is also a module for Iptables that allows you to limit how many alerts are logged so that you do not flood your logs with huge amounts of duplicate data, when a host scan is performed for instance. See the attached firewall configuration script for more information on configuring the firewall rule sets.

This open source firewall solution will save your company approximately \$26,505. The total cost for the open source solution is broken down below (See Figure 8). The equivalent for a popular closed source solution, Checkpoint-1, is also provided.

	<u>Open Source</u>	<u>Closed Source</u>
Server	\$2,500.00	\$2,500.00
Software	\$0.00	\$22,670.00
Installation*	\$200.00	\$2,000.00
Support**	\$2,600.00	\$3,735.00
Training***	\$400.00	\$2,500.00
Bottom Line	\$5,700.00	\$33,405.00
Difference		\$27,705.00

Figure 8. Price Comparison

To provide redundancy in your bridging firewall it is possible to have as many NICs as your server will support. If one NIC should fail, the bridge will use the other NICs and your local hosts will not notice any disconnection from the Internet. Redundancy can be taken a step further by having another bridging firewall in parallel with the first (See Figure 8). This allows one bridge to go down and the other one will pick up. This configuration also provides some load balancing.

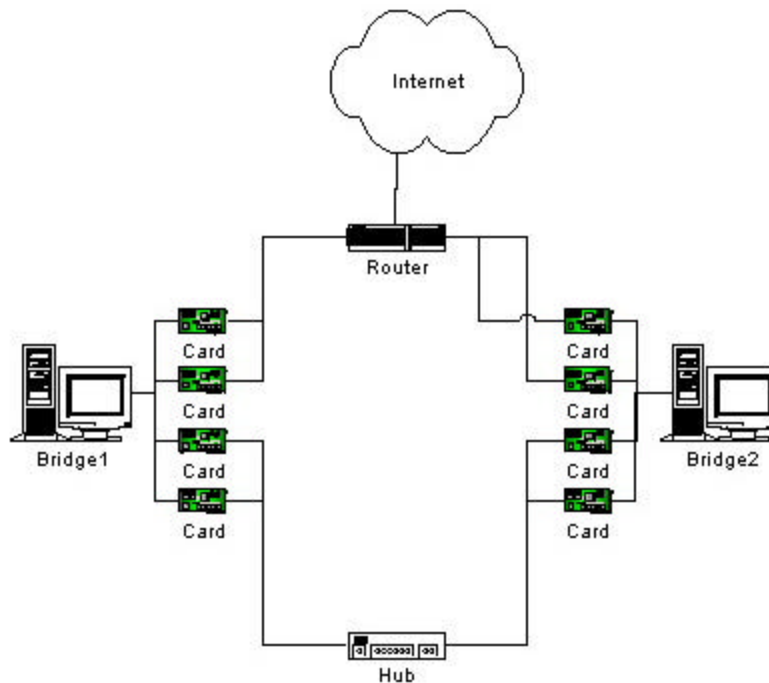


Figure 8. Dual bridges with multiple NICs for optimal redundancy.

*Installation for Open Source is based on IT professional making \$100/hour; Closed Source is based on installation by company providing product.

**Support for Open Source is based on \$50/week to update firewall rulesets and monitoring logs. Closed Source is based on \$50/week to update firewall rulesets as well as \$1,135/year support contract.

***Training for Open Source is based on IT professional making \$100/hour and time it takes to read manuals; Closed Source is based on Checkpoint Certification courses

Attachment 1.

Below is a script to be placed in the `/etc/rc.d/init.d/` directory. It will be started whenever the firewall is booted up. This script will also allow you to send the following commands:

```
/etc/rc.d/init.d/bridge start - starts the bridge
```

```
/etc/rc.d/init.d/bridge stop - stops the bridge
```

```
/etc/rc.d/init.d/bridge restart - stops and starts the bridge
```

```
/etc/rc.d/init.d/bridge status - executes brctl show bridgename
```

```
#!/bin/bash
# /etc/rc.d/init.d/bridge
#

return=$rc_done
case "$1" in

    start)
        echo "Starting service bridge mybridge"
        brctl addbr mybridge || return=$rc_failed
        brctl addif mybridge eth0 || return=$rc_failed
        brctl addif mybridge eth1 || return=$rc_failed
        ifconfig eth2 0.0.0.0 promisc || return=$rc_failed
        ifconfig eth3 0.0.0.0 promisc || return=$rc_failed
        brctl sethello mybridge 1 || return=$rc_failed
        brctl setmaxage mybridge 4 || return=$rc_failed
        brctl setfd mybridge 4 || return=$rc_failed
        ifconfig mybridge promisc up || return=$rc_failed
        echo -e "$return"
        ;;

    stop)
        echo "Shutting down service bridge mybridge"
        brctl delif mybridge eth0 || return=$rc_failed
        brctl delif mybridge eth1 || return=$rc_failed
        brctl delbr mybridge || return=$rc_failed

        echo -e "$return"
        ;;

    status)
        ifconfig mybridgeb
        brctl show mybridge
        ;;

```

```
restart)
    $0 stop && $0 start || return=$rc_failed
    ;;

*)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac

test "$return" = "$rc_done" || exit 1
exit 0
```

Attachment 2.

This is a script that configures the firewall with a defined rule set. Read the comments in the script for explanations on the different rules.

```
#Flush all rules from the chains
iptables -F
#Delete all user created chains (mainly KEEP_STATE chain)
iptables -X

#####
##Create special chain KEEP_STATE
iptables -N KEEP_STATE
iptables -F KEEP_STATE
##Drop bad states
iptables -A KEEP_STATE -m state --state INVALID -j DROP
iptables -A KEEP_STATE -m state --state RELATED,ESTABLISHED -j ACCEPT
#####

#Deny bad packets
iptables -A FORWARD -p tcp --tcp-flags ALL FIN,URG,PSH -m limit --limit 5/minute -j LOG -
-log-level notice --log-prefix "NMAP-XMAS: "
iptables -A FORWARD -p tcp --tcp-flags ALL FIN,URG,PSH -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit --limit 5/minute -j LOG -
-log-level notice --log-prefix "SYN/FIN: "
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -m limit --limit 5/minute -j LOG -
-log-level notice --log-prefix "SYN/RST: "
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP

#Drop RST/ACKs to limit OS detection through ping
iptables -A FORWARD -p tcp --tcp-flags RST RST,ACK -m limit --limit 5/minute -j LOG --
log-level notice --log-prefix "RST/ACK: "
iptables -A FORWARD -p tcp --tcp-flags RST RST,ACK -j DROP

#Drop possible compaq directory traversal port
iptables -A FORWARD -p tcp --dport 2301 -j DROP

#Deny pings from outside
iptables -A FORWARD -p icmp --icmp-type 0/0 -d 192.168.1.0/24 -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type 0/0 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Drop Echo Reply: "
iptables -A FORWARD -p icmp --icmp-type 0/0 -j DROP

#Drop potential SQL Worm
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 1433 -j ACCEPT
iptables -A FORWARD -p tcp --dport 1433 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Possible SQL Worm: "
iptables -A FORWARD -p tcp --dport 1433 -j DROP

#Pass all boxes to the keep_state chain
iptables -A FORWARD -j KEEP_STATE

#####
##Set up UDP
#Outgoing Traceroute
iptables -A FORWARD -p udp -s 192.168.1.0/24 --sport 32769:65535 --dport 33434:33523 -j
ACCEPT

#Incoming Traceroute
iptables -A FORWARD -p udp -s 192.168.1.0/24 --dport 32769:65535 --sport 33434:33523 -j
ACCEPT

#Time exceeded
iptables -A FORWARD -p udp -s 192.168.1.0/24 --dport 11 -j ACCEPT
```

```

#Port not found
iptables -A FORWARD -p udp -s 192.168.1.0/24 --dport 3 -j ACCEPT

#DNS
iptables -A FORWARD -p udp -s 192.168.1.0/24 --dport 53 -j ACCEPT

#DHCP
iptables -A FORWARD -p udp -s 192.168.1.0/24 --sport 68 --dport 67 -j ACCEPT

#Time Server
iptables -A FORWARD -p udp -s 192.168.1.0/24 --sport 1024:65535 --dport 123 -j ACCEPT

#####

##Allow outward browsing
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 80 -j ACCEPT

##Allow outward ssh
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 22 -j ACCEPT

##Allow outward ftp
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 21 -j ACCEPT

##Allow outward telnet
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 23 -j ACCEPT

##Allow outward smtp
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 25 -j ACCEPT

##Allow outward pop
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 110 -j ACCEPT

##Set up web servers to allow access to from outside
#www
iptables -A FORWARD -p tcp -d 192.168.1.100 --dport 80 -j ACCEPT

###Block HTTP Request from outside that are not authorized
iptables -A FORWARD -p tcp --dport 80 -j DROP
iptables -A FORWARD -p tcp --dport 80 -m limit --limit 5/minute -j LOG --log-level notice
--log-prefix "Dropped HTTP: "

#Allow VNC out
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 5900 -j ACCEPT
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 5800 -j ACCEPT

#Allow Telnet out
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 23 -j ACCEPT

#Deny ports
##telnet
iptables -A FORWARD -p tcp --dport 23 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Denied Telnet: "
iptables -A FORWARD -p tcp --dport 23 -j DROP

##VNC
iptables -A FORWARD -p tcp --dport 5800 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Denied VNC: "
iptables -A FORWARD -p tcp --dport 5800 -j DROP

##VNC
iptables -A FORWARD -p tcp --dport 5900 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Denied VNC: "
iptables -A FORWARD -p tcp --dport 5900 -j DROP

##Deny BO
iptables -A FORWARD -p udp --dport 31337 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Denied BO: "
iptables -A FORWARD -p udp --dport 31337 -j DROP

##FTP
#Allow ftp on 192.168.1.101
iptables -A FORWARD -p tcp -s 192.168.1.101 --dport 21 -j ACCEPT

```

```
#Deny and log ftp on all others
iptables -A FORWARD -p tcp -s 192.168.1.0/24 --dport 21 -j DROP
iptables -A FORWARD -p tcp --dport 21 -m limit --limit 5/minute -j LOG --log-level
notice --log-prefix "Denied FTP: "

#Deny suspicious traffic from wani3.he.net
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 1080 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 8080 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 8000 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 80 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 3128 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 81 -j DROP
iptables -A FORWARD -p tcp -s 64.71.137.42 --destination-port 8081 -j DROP

##Allow all outgoing traffic
iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT

#Set to drop all packets not accepted by rules above
iptables -A FORWARD -j DROP
```

References

Australian Computer Emergency Response Team. Unix Security Checklist 2.0.
http://www.uscert.org.au/Information/Auscert_info/Papers/usc20.html.

Böhme, Uwe. <http://www.linuxdoc.org/HOWTO/BRIDGE-STP-HOWTO/>.

Breuer, Peter. <http://www.linuxdoc.org/HOWTO/mini/Bridge+Firewall.html>.

Buytenhek, Lennert. <http://bridge.sourceforge.net>.

Chapman, D. Brent. Building Internet Firewalls. 1995. O'Reilly.

Ziegler, Robert L. Linux Firewalls. 2000. New Riders.