# White Paper: Indestructible Firewall In A Box v1.0

Nick McCubbins

## 1.1 Credits

- Nathan Yawn (yawn@emacinc.com)

## 1.2 Acknowledgements

- Firewall-HOWTO
- Linux Router Project
- LEM

## 1.3 Revision History

- Version 1.0 First public release

## 1.4 Feedback

- Send all information and/or criticisms to pmccubbins@emacinc.com

## 1.5 Distribution Policy

## 2 Abstract

In this document, the procedure for creating an embedded firewall whose root filesystem is loaded from a flash disk and then executed from a RAMdisk will be illustrated. A machine such as this has uses in many environments, from corporate internet access to sharing of a cable modem or xDSL connection among many computers. It has the advantages of being very light and fast, being impervious to filesystem corruption due to power loss, and being largely impervious to malicious crackers.

The type of firewall illustrated herein is a simple packet-filtering, masquerading setup. Facilities for this already exist in the Linux kernel, keeping the system's memory footprint small. As such the device lends itself to embedding very well. For a more detailed description of firewall particulars, see the Linux Firewall-HOWTO.

## 3 Equipment

This project has minimal hardware requirements. An excellent configuration consists of:

For a 100-baseT network:

- SBC-554 Pentium SBC with PISA bus and on-board PCI NIC (http://www.emacinc.com/pc.htm#pentiumsbc), approx. $373
- PISA backplane, chassis, power supply (http://www.emacinc.com/sbcpc_addons/mbpc641.htm), approx. $305

- Second PCI NIC
  - 32 MB RAM
  - 4 MB M-Systems Flash Disk (minimum), approx. $45

  For a 10-baseT network:

  - EMAC's Standard Server-in-a-Box product (http://www.emacinc.com/server_in_a_box.htm), approx. $589
  - Second 10-baseT Ethernet option (http://www.emacinc.com/sbc_pc_addons.htm#pc104), approx. $108

This system, composed of dedicated embedded components, will generally have lower power consumption, and be far more resistant to shock, vibration, and temperature variation than standard systems. It will also require far less space than a conventional desktop or tower case. Finally, these components are usually available for several years (as opposed to a few months for commodity hardware), making it easy to create, maintain, and support a single product. Also, keep in mind that EMAC's embedded Linux distribution can be ordered pre-configured as a firewall, which will save you configuration time.

A testbed / budget system may be built using the following:

- Pentium-class system (60 Mhz or better)
- 32 MB RAM
- Two Kingston Realtek 8139-based PCI 10/100 NICs
- PCD-897 DiskOnChip 2000 ISA adapter (approx. $32)
- 4MB M-Systems Flash Disk

A setup such as this can serve a full-size ethernet network from a T1 internet connection. Firewalling consumes almost no CPU resources, but does necessitate having a PCI bus for 100Mb ethernet cards. A 25 MHz 386 is sufficient for cable modems and the like.

Notice there is no mention of a hard disk, video card, or keyboard. These are unnecessary after successful configuration. The firewall loads its entire file system from DiskOnChip Flash Disk, eliminating the need for further mass storage. The system is also extremely reliable, and easy to maintain: if a problem should arise, the reset button is the only remedy required. This eliminates the need for both video and keyboard.

## 3.1 Development Requirements

- Working Linux system for creating initial ramdisk

## 3.2 Software Needed

- Syslinux boot loader

- Linux kernel with loopback FS support (on development system)
- Embedded Linux distribution (EMAC, LEM, LRP, etc.)

## 3.3 Supplemental Documentation

- Loopback FS HOWTO: http://www.linux.org/docs/ldp/howto/mini/Loopback-Root-FS-3.html
- Firewall and Proxy Server HOWTO: http://www.linux.org/docs/ldp/howto/Firewall-HOWTO-6.html
- LEM Embedded Linux Distribution: http://www.linux-embedded.com/lem.php3

## 4. Setup

## 4.1 Selecting An Embedded Distribution

First, if not using EMAC's embedded Linux distribution you will need to choose one of the many embedded Linux distributions available. Peruse http://www.linux-embedded.com or http://www.freshmeat.net for a list.

## 4.2 Setting Up the Ramdisk

In order to create a ramdisk image, your host system must have loopback filesystem support in the kernel (`CONFIG_BLK_DEV_INITRD`). The process for creating the initial ramdisk is not unlike the procedure for making a boot floppy's initial ramdisk; the principal difference is that you'll have more space. Keep in mind, however, that the bigger the ramdisk, the more memory the firewall will require. A ramdisk of 4MB should be sufficient in building the setup.

To create the ramdisk, issue:

```
# dd if=/dev/zero of=ramdisk.img bs=1k count=4096
```

This command will create a 4MB file in the current directory. This file will become the ramdisk image. Format this image:

```
# mke2fs -m0 -F ramdisk.img
```

Mount the ramdisk:

```
# mount -o loop ramdisk.img /mnt
```

Now, unpack the embedded distribution you chose, and copy all necessary files into `/mnt`. If you use LEM, you unpack the distribution directly into `/mnt`. Other distributions may be larger, and may require you to remove unneeded binaries and components before copying them into the ramdisk image. Keep in mind that you are creating an entire Linux system in this ramdisk image, so be sure you have all the files and shared libraries that the system needs to run!

Once you have finished creating a minimal Linux system in `/mnt`, you will need to configure basic networking, hosts, routes, etc. Methods for this vary from one embedded distribution to the next -- read the documentation that came with your distribution and the Firewall-HOWTO for exact details. One ethernet card will need to be configured with a valid internet IP address and network setup, and the other will need to be configured for your "inside network," `10.0.2.x` in our case.

One important note: Be sure that `/dev/ram` is block major 1 and minor 0. A lot of distributions erroneously symlink `/dev/ram` to `/dev/ram1`. Make sure both the minimal system in `/mnt` and your development machine are correct.

The ipchains utility is the core command for the firewalling setup; if it does not come with the embedded distribution you chose, you can copy it from your development system. A sample configuration script for the firewalling code would look something like this:

```
#!/bin/bash
/sbin/ipchains -A input -s 10.0.2.0/24 -j MASQ
echo "1" > /proc/sys/net/ipv4/ip_forward
```

One rule is all you need for a basic setup. The above script assumes that all of the computers on your local network have private IP addresses `10.0.2.2`, `10.0.2.3`, etc. This script should be placed in a directory where scripts are run at boot time, e.g. `/etc/rc.boot`.

When you're finished configuring the filesystem, unmount the ramdisk and GZIP it:

```
# umount /mnt
# gzip -9 ramdisk.img
```

## 4.3 Creating the Firewall Kernel

A monolithic kernel is recommended (no module support) for this project, for reliability and ease-of-use. The kernel will not be added to the filesystem, but will reside separately on a DOS partition (more later). Using a monolithic kernel will allow for easy kernel upgrades when required.

To enable firewalling, the networking options section should look similar to this:

```
<*> Packet socket
[ ] Kernel/User netlink socket
[*] Network firewalls
[ ] Socket Filtering
<*> Unix domain sockets
[*] TCP/IP networking
[ ] IP: multicasting
[*] IP: advanced router
[ ] IP: kernel level autoconfiguration
[*] IP: firewalling
[*] IP: always defragment (required for masquerading)
[*] IP: transparent proxy support
[*] IP: masquerading
--- Protocol-specific masquerading support will be
built as modules.
[*] IP: ICMP masquerading
--- Protocol-specific masquerading support will be
built as modules.
[ ] IP: masquerading special modules support
[*] IP: optimize as router not host
< > IP: tunneling
< > IP: GRE tunnels over IP
[ ] IP: aliasing support
[ ] IP: TCP syncookie support (not enabled per default)
--- (it is safe to leave these untouched)
< > IP: Reverse ARP
[*] IP: Allow large windows (not recommended if <16Mb
of memory)
< > The IPv6 protocol (EXPERIMENTAL)
---
< > The IPX protocol
< > Appletalk DDP
< > CCITT X.25 Packet Layer (EXPERIMENTAL)
< > LAPB Data Link Driver (EXPERIMENTAL)
[ ] Bridging (EXPERIMENTAL)
[ ] 802.2 LLC (EXPERIMENTAL)
< > Acorn Econet/AUN protocols (EXPERIMENTAL)
< > WAN router
[ ] Fast switching (read help!)
[ ] Forwarding between high speed interfaces
[ ] CPU is too slow to handle full bandwidth
```

When the kernel finishes compiling, it will be the file `/usr/src/linux/arch/i386/boot/bzImage`. Use the `RDEV` command to change the kernel's root filesystem to `/dev/ram0`:

```
# rdev /usr/src/linux/arch/i386/boot/bzImage /dev/ram0
```

## 4.4 Creating the boot device

Using `fdisk`, create a DOS partition on the boot media, and set the bootable flag. This partition must be large enough to hold both the ramdisk image and the kernel. If you are using DiskOnChip, this partition should be the entire DoC. Once you have saved the partition table, you need to format the partition and install Syslinux:

```
# mkfs.dos <device>

# syslinux <device>
```

Rename the kernel image to `linux`, then copy it and the ramdisk image to this bootable partition.

Create a file called `syslinux.cfg` on the bootable partition which contains the line:

```
append load_ramdisk=1 initrd=ramdisk.img.gz
```

This will cause the syslinux bootloader to tell the kernel that an initial ramdisk is present at boot time.

## 4.5 Boot sequence

Install the DoC in the target system, and power it up. You may need to adjust the target system's BIOS settings to scan for BIOS extensions (also called boot ROMs). You should also disable BIOS caching for the area of memory where your DiskOnChip will reside (check the jumpers on the DoC adapter card to find where this will be). Connect the two ethernet ports to the correct networks...this may require a bit of experimentation to determine which card is set for which network.

For debugging purposes, it is best to plug in a monitor and watch the system boot the first time. The boot sequence is as follows:

1. BIOS
2. DoC boot sector (syslinux)
3. Ramdisk loaded
4. Kernel loaded
5. Control passed to kernel, kernel informed of ramdisk location
6. Kernel uncompresses self, boots
7. Kernel mounts ramdisk
8. ramdisk is re-mounted as root
9. boot scripts are run
10. login prompt/boot process finished

## 5 Setting Up Clients

In order to set up client workstations to use your newly created firewall, simply give them an address on the private subnet (the `10.0.2.x` network) and set their default gateway to the firewall's internal IP address. On Microsoft Windows clients, this is done using the appropriate tab in the Network Settings control panel applet. Under Linux, the process varies by distribution, and may include editing boot scripts or using the `linuxconf` program. If you can contact hosts on the internet after this configuration you are finished.

## 6 Troubleshooting

You should start troubleshooting by ensuring that your client machines can ping the inside address of the firewall. If not, check your cabling from the clients to the firewall. Next, try pinging a server at your ISP, using one of the client machines. If this fails, check your gateway settings on the client machines. If this succeeds, double-check with your ISP that they are routing packets to your firewall's outside IP.