LINUX JOURNAL

Building a Diskless 2.6 Firewall

By Christian Herzog
Created 2004-08-25 01:00

For your next DIY project, pick up an old Pentium computer and a CompactFlash card and build a custom router/firewall.

Want to build a custom router/firewall for your home network? You can obtain the necessary hardware virtually for free from garage sales or on-line auctions. You even might have some old hardware lying around. A Pentium-class system is more than sufficient and can handle the stress well. Typically, we don't need much memory, but I recommend at least 16MB of RAM. In place of a hard disk, we can use a compact Flash, or CF, card. CF has some nifty features, such as on-board error detection and correction to minimize Flash wear. Due to a full-fledged IDE interface, it also can be used as a normal IDE device. You do need an adapter to connect the card, though. We are going to use only two to three megabytes, so the size of the card doesn't really matter.

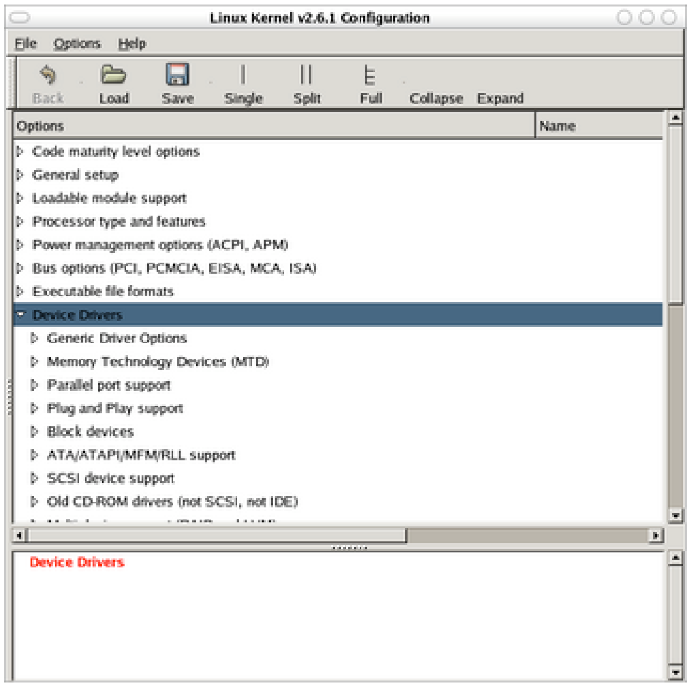
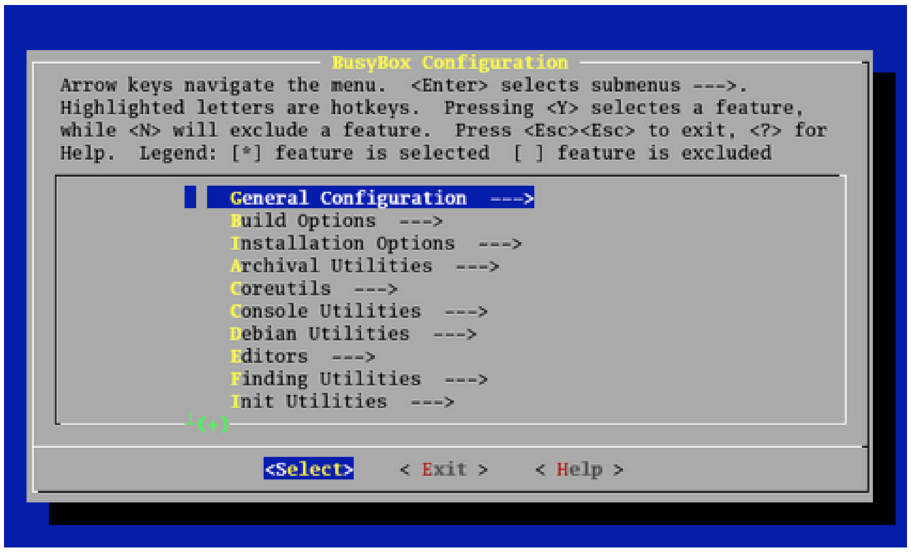

Linux Kernel v2.6.1 Configuration
File
Options
Help
Back
Load
Save
Single
Split
Full
Collapse
Expand
Options
Name
Code maturity level options
General setup
Loadable module support
Processor type and features
Power management options (ACPI, APM)
Bus options (PCI, PCMCIA, EISA, MCA, ISA)
Executable file formats
Device Drivers
Generic Driver Options
Memory Technology Devices (MTD)
Parallel port support
Plug and Play support
Block devices
ATA/ATAPI/MFM/RLL support
SCSI device support
Old CD-ROM drivers (not SCSI, not IDE)
Device Drivers
[1]

Figure 1. The GTK+ Version of the Kernel Configuration Tool



```
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help.  Legend: [*] feature is selected  [ ] feature is excluded
```

[2]

Figure 2. The ncurses-Based Configuration of BusyBox

Step One, Getting the Software

For now, we are going to build a machine with the following features: iptables firewall, SSH dæmon, DHCP server and DNS server. Because we're going to build a Linux system completely from scratch, we need a fair amount of software. The usual Linux tools aren't built for embedded systems, they're loaded with features we don't need. This is where BusyBox, the Swiss Army knife of embedded Linux, comes into play. We can exchange most of our needed tools with BusyBox, for instance a shell, ifconfig, ip tools and so on. We then need a bootloader, a C library, iptables, an SSH server and a DNS server. We are going to use the new 2.6.1 kernel release, which introduces some issues we'll deal with later.

I use GRUB as the bootloader, but any recent loader should do. iptables, the 1.2.9 version, is the choice for our firewalling software, and Dropbear serves as our SSH dæmon. Finally, we use the handy dnsmasq program, basically a stripped down yet fully functional DNS server to forward our requests to your ISP's DNS servers.

The biggest problem I faced was the C library, libc. I opted for the lightweight library uClibc. It's a C library for embedded systems that comes with a whole toolchain. The development environment can be installed easily by getting the installer tar file from http://uclibc.org/cgi-bin/cvsweb/toolchain/gcc-3.3.x.tar.gz?view=tar [3]. Unpack it and edit the Makefile in the toolchain directory, then type `make`. This downloads, compiles and installs a toolchain for your platform. It takes some time to install it, mostly depending on your Internet connection. After it's finished you are presented with a directory named toolchain_ARCHITECTURE (for example, toolchain_i386) containing all libraries, header files and a cross-compiler needed later.

Step Two, Preparing Our CF Card

One of the biggest questions is "What filesystem should I use?" Compact Flash cards are shipped formatted with VFAT, which probably isn't the best choice for Linux. The obvious choice would be ext3, but you should be concerned about the Flash wear that affects all Flash-based devices sooner or later. All Flash-based media fails after a certain amount of write cycles; compact Flash usually fails at around a million. That might sound like a lot, but the journaling function of the filesystem would kill the Flash eventually. For simplicity, we use the ext2 filesystem, which we mount read-only. To configure the router, you have to remount the disk writeable.

Hook up the card to a USB reader or use an IDE adapter, and use your favourite fdisk program to create a new partition. One partition the size of your Flash should suffice. Then issue the command `/sbin/mkfs.ext2 -m0 /dev/[flash]` to create a new ext2 filesystem.

Issue the command `tune2fs -c0 -i0 /dev/[flash]` to turn off the automatic filesystem checking and the warning that the filesystem should be checked for errors. Because we use it read-only, we don't need to worry about errors too much. Mount the disk and proceed.

Step Four, Compile the Kernel

The new kernel has seen some major improvements, the first and most obvious being the new graphical configuration menu. It is invoked by either `make gconfig` or `make xconfig` for the GTK+ or Qt-based application, respectively (Figure 1). Using these new GUIs, configuring the kernel is a snap, easier than ever before.

One thing you might want to consider when compiling the kernel for the router is that BusyBox's module support appears to be somewhat flaky with the new 2.6 modules. We can dodge that problem easily by compiling a non-modular kernel that offers the nice side-effect that we can forget about the module-related utilities.

Feature-wise I advise you to check all iptables-related options and the driver for your network cards. We can leave most other options unchecked unless you need them. Remember, the less we choose to add, the less space the kernel needs. Be sure to compile all options statically, though, because we aren't using the modules.

When finished, enter `make` and watch the new, tidied compilation process. Create a directory boot on your compact Flash and `cp` the kernel from arch/i386/boot/ to /mnt/cf/boot/.

Step Five, Compiling the Applications

Compiling the applications is quite easy; we start with BusyBox. Unpack it, change to the new directory and type `make menuconfig`. You are presented with a simple and clean ncurses-based configuration menu (Figure 2). Press B for the Build Options and enter the prefix of the uClibc toolchains GCC program. I copied my toolchain_i386 directory to /usr, so the full path I have to enter is /usr/toolchain_i386/bin/i386-linux-. Next, go to Installation Options and change your installation directory to the mount point of your Flash card. Activate the ifconfig, inetd, ip and udhcpd options and all sub-options. You should find a well-commented configuration file for udhcpd in BusyBox's examples directory. Finally, go to the Init Utilities section and confirm that all options are checked. You might want to add vi as well as other tools to increase functionality.

BusyBox can provide you with tools for login and user management. Nevertheless, I somehow wasn't able to get them to work properly. I therefore had to use the TinyLogin package from http://tinylogin.busybox.net [4]. Configuration and installation is quite simple; simply edit the Makefile to use the cross compiler and install as user root, so TinyLogin can have the SUID bit set.

IPv6 isn't supported in this configuration. For sure, it is a feature to add in the future, but IPv6 is rather uncommon for typical SOHO environments. The lack of IPv6 support leaves us with the need to disable it in our applications.

Although it's quite easy to disable this feature on iptables (`make DO_IPV6=0 KERNEL_DIR=[path your Linux kernel] CC=[path to your cross compiler]`), dnsmasq wouldn't be convinced to drop this functionality. After some hours of serious head scratching, I tried the latest unstable version of the little DNS dæmon, namely 2.0rc1. The command `make CC=[path to cross compiler]` worked instantly and produced a perfectly working dnsmasq binary. The 2.0 versions also have an embedded DHCP dæmon, which one might favour over the one that ships with BusyBox.

You should use the strip command on all binaries to get rid of debugging information and make them smaller.

The dropbear SSH dæmon works quite nicely and compiles without modifications using uClibc. If you run into problems, consult the INSTALL file, which also explains how to configure it. Copy the resulting binaries (dropbear, dropbearkey) to /sbin.

Apart from that, you need to add the line:

```
none  /dev/pts  devpts  gid=5,mode=620  0 0
```

to the file /etc/fstab, so the openpty() function of dropbear works and you actually can log in. Apart from that, you need the directory /dev/pts and the device file /dev/ptmx. /dev/urandom is needed to generate the RSA and DSS keys by way of the dropbearkey program.

Step Six, Filling the Filesystem

Change to /mnt/cf. After the installation of BusyBox and TinyLogin, we're already welcomed with a somewhat familiar environment. You now have to add the following directories: /dev, /etc, /lib and /proc.

Go to the directory /lib and copy the following libraries and symbolic links from your toolchain_i386/lib directory. It should look like Listing 1.

Listing 1. Libraries and Symbolic Links to Copy

```
$ ls -lh
total 295K
-rwxr-xr-x    1 root     root          16K Jan 18 17:40 ld-uClibc-0.9.26.so
lrwxr-xr-x    1 root     root           19 Jan 19 13:33 ld-uClibc.so.0 -> ld-uClibc-0.9.26.so
-rw-r--r--    1 root     root         8.9K Jan 18 17:40 libcrypt-0.9.26.so
lrwxr-xr-x    1 root     root           13 Jan 19 13:33 libcrypt.so -> libcrypt.so.0
lrwxr-xr-x    1 root     root           18 Jan 19 13:33 libcrypt.so.0 -> libcrypt-0.9.26.so
lrwxr-xr-x    1 root     root           19 Jan 19 13:33 libc.so.0 -> libuClibc-0.9.26.so
-rw-r--r--    1 root     root         5.6K Jan 18 17:40 libdl-0.9.26.so
lrwxr-xr-x    1 root     root           10 Jan 19 13:33 libdl.so -> libdl.so.0
lrwxr-xr-x    1 root     root           15 Jan 19 13:33 libdl.so.0 -> libdl-0.9.26.so
-rw-r--r--    1 root     root         1.5K Jan 18 17:40 libnsl-0.9.26.so
lrwxr-xr-x    1 root     root           11 Jan 19 13:33 libnsl.so -> libnsl.so.0
lrwxr-xr-x    1 root     root           16 Jan 19 13:33 libnsl.so.0 -> libnsl-0.9.26.so
-rw-r--r--    1 root     root         255K Jan 18 18:16 libuClibc-0.9.26.so
-rw-r--r--    1 root     root         4.2K Jan 18 17:40 libutil-0.9.26.so
lrwxr-xr-x    1 root     root           12 Jan 27 16:02 libutil.so -> libutil.so.0
lrwxr-xr-x    1 root     root           17 Jan 27 16:02 libutil.so.0 -> libutil-0.9.26.so
```

Copy the iptables and dnsmasq binaries to /sbin, and switch to the /dev directory. Copy the devices shown in Listing 2 from your /dev directory. Use the command **`cp -dpr`** to copy the device files, otherwise they might not work. Of course, you also can use the mknod command to create the devices from scratch.

Listing 2. Devices to Copy

```
crw-------    1 root     root       5,   1 Jan 17 15:34 console
crw-------    1 root     root       1,   6 Sep 15 15:40 core
brw-rw----    1 root     disk       3,  64 Sep 15 15:40 hdb
```

```
brw-rw----    1 root     disk       3,  65 Sep 15 15:40 hdb1
crw-rw-rw-    1 root     root       1,   3 Sep 15 15:40 null
crw-rw-rw-    1 root     root       5,   2 Jan 27 17:32 ptmx
drwxr-xr-x    2 root     root         1024 Jan 20 15:15 pts
crw-r--r--    1 root     root       1,   8 Sep 15 15:40 random
lrwxrwxrwx    1 root     root           17 Jan 18 15:25 stderr -> ../proc/self/fd/2
lrwxrwxrwx    1 root     root           17 Jan 18 15:25 stdin -> ../proc/self/fd/0
lrwxrwxrwx    1 root     root           17 Jan 18 15:25 stdout -> ../proc/self/fd/1
crw-rw-rw-    1 root     root       5,   0 Sep 15 15:40 tty
crw--w----    1 root     root       4,   0 Sep 15 15:40 tty0
crw-------    1 root     root       4,   1 Jan 17 15:35 tty1
crw-------    1 root     root       4,   2 Jan 17 15:35 tty2
crw-------    1 root     root       4,   3 Jan 17 15:35 tty3
crw-------    1 root     root       4,   4 Jan 17 15:35 tty4
crw-------    1 root     root       4,   5 Jan 17 15:35 tty5
crw-r--r--    1 root     root       1,   9 Jan 27 15:57 urandom
crw-rw-rw-    1 root     root       1,   5 Sep 15 15:40 zero
```

Now, we're going to add the user root. Issue the command **`/usr/sbin/chroot ./ /bin/ash`**, and you are presented with BusyBox's ash shell. Go to the directory /etc and create the empty files passwd, shadow and group.

With the command **`adduser -h /root root`** you can add the user root. You have to edit the file /etc/passwd and change the line **`root:x:500:500:Linux User,,,:/root:/bin/sh`** to **`root:x:0:0:Linux User,,,:/root:/bin/sh`** in order to make the user root the superuser. Now type **`su root`** and create a password with the command **`passwd`**.

After that we need to add the usual configuration and system files: hosts, securetty, fstab, inittab and resolv.conf. I didn't go for the full-blown init scripts, my inittab consists of only two lines:

```
tty1::respawn:/sbin/getty 38400 tty1
::sysinit:/etc/rc
```

All applications are launched with the rc script (Listing 3). The network interfaces also are configured directly with this script.

Listing 3. rc Script

```
#!/bin/sh
/bin/mount -a

echo "Setting up network"
/sbin/ifconfig lo 127.0.0.1
/sbin/ifconfig eth0 192.168.0.1
/sbin/ifconfig eth1 10.0.0.2
/sbin/route add default gw 10.0.0.1
echo "Starting daemons"
/sbin/inetd
/sbin/dropbear
/sbin/udhcpd
/sbin/dnsmasq

Starting Firewall
/etc/firewall.sh
```

I created the firewall.sh script with Fwbuilder, which is by far the easiest and most comfortable solution for getting a firewall up and running. For more information on Fwbuilder, read Mick Bauer's articles from the May and June 2003 issues of *Linux Journal*. If you have to change the script, you need to log in using SSH and use the command **`mount -o rw,remount /dev/hda1 /`** to make the Flash card writeable. Then, transfer the script from your machine to the firewall and move it to /etc/firewall.sh. Finally, make the script executable and use the command **`mount -o ro,remount /dev/hda1 /`** to make the filesystem read-only again. You might want to automate this process by writing a little shell script.

System Logging

One good solution for system logging would be to send the log messages to another machine over the network. This option works well but requires another computer; nevertheless, it is my choice. A different approach would require another partition, mounted read-write, to log to, which strains the Flash quite a bit.

Final Step, Installing the GRUB Bootloader

The 2.6 series kernels require a proper bootloader. Simply writing the kernel directly onto the disk is deprecated and is quite messy after all. Create a directory called grub in /mnt/cf/boot and copy stage1, stage2 and e2fs_stage_1_5 from your /boot/grub directory. Apart from that, you need a configuration file for GRUB called menu.lst. As a side note, I had to create a symlink named grub.conf, otherwise GRUB wouldn't boot-this appears to be a Red Hat-specific glitch. menu.lst needs to be dropped in /boot/grub.

Next, you need a boot floppy created with the command **`cat stage1 stage2 > /dev/fd0`**. Boot from the floppy, and you are dropped to the GRUB shell. Type **`root (hd0,0)`** and **`setup (hd0)`**. GRUB spills out some debug messages that should indicate whether the procedure was successful. Eject the floppy and type **`reboot`**. The Firewall should boot now, and after a few seconds you should be presented with the login prompt.

Things You Might Want to Add

We now have a very basic Linux system geared toward firewalling and routing. You easily can add the PPP dæmon for dial-up style broadband Internet access. You also could add a printing dæmon and spool print jobs to memory. The 2.6 kernel also has a native ipsec implementation, compile it into the kernel and add the user-space tools for full VPN ability. As you can see, the possibilities are virtually endless.

The *Linux Journal* FTP site has a tarball of the example filesystem from this article (see Resources), which might help if you run into trouble.

Resources

BusyBox: http://www.busybox.net [5]

Dropbear: http://matt.ucc.asn.au/dropbear/dropbear.html [6]

Example Filesystem: ftp://ftp.ssc.com/pub/lj/listings/Web/7383.tar [7]

TinyLogin: http://tinylogin.busybox.net [8]

uClibc: http://www.uclibc.org [9]

"Using Firewall Builder, Part I": http://www.linuxjournal.com/article/6625 [10]

"Using Firewall Builder, Part II": http://www.linuxjournal.com/article/6715 [11]

Links

[1] http://www.linuxjournal.com//articles/web/2004-08/7383/7383f1.png
[2] http://www.linuxjournal.com//articles/web/2004-08/7383/7383f2.png
[3] http://www.linuxjournal.com/
[4] http://www.linuxjournal.com/
[5] http://www.busybox.net
[6] http://matt.ucc.asn.au/dropbear/dropbear.html
[7] ftp://ftp.ssc.com/pub/lj/listings/Web/7383.tar
[8] http://tinylogin.busybox.net
[9] http://www.uclibc.org
[10] http://www.linuxjournal.com//article/6625
[11] http://www.linuxjournal.com//article/6715

Source URL: http://www.linuxjournal.com/article/7383