

Building A WLAN Proxy Server Part 2

Michael J. Martin

What is a Kernel?

In essence, the kernel is the Unix/Linux operating system. The kernel controls the hardware, I/O, memory management, system and user processes. On whole, it manages all the critical aspect of the computer necessary for operation. Applications, such as shells, services like sshd, text editors and the like all operate "around" the kernel, which allocates the processor and memory resources the programs need to function, hence the name kernel. All Linux distributions come with a kernel. But, like the OpenSSL and OpenSSH distributions, the kernel on the Linux distribution is often behind the current build. Each new kernel build or sub-build adds new feature and driver support, along with correcting any bugs or security vulnerabilities. We are building a kernel to ensure that the networking support we need to support our proxy server is in place. For some, building a kernel is a somewhat daunting task, this was true with earlier versions of the Linux kernel. Since the release of the 2.4 kernel, which is now essentially "General Deployment," there are a number of resources on Web on how to build a Linux kernel. Our goal is cover all of the elements you need to include in the kernel build to support the proxy server environment.

There are three basic steps to building the kernel: Prep, compile and install.

Before starting the kernel build, you need to get some basic information on the server your building the kernel for. There are two resources you can use to gather this data. The first is the output `/bin/dmesg` that prints the output of the kernel load process, allowing you to see which drivers were loaded by the kernel that is currently running. The `/bin/dmesg` output is sent right to command shell; to save it to a text file, you can redirect the output. Here is the syntax to save the output to a file and then print the output to the shell:

```
/bin/dmesg > harddep.txt; more harddep.txt
```

Here are some other command output options you can use with `/bin/dmesg` to collect specific information.

- To determine the Ethernet hardware: `/bin/dmesg | more | grep eth`
- To determine the root partition: `/bin/dmesg | more | grep "/dev/hd"`
- To determine the file-system type: `/bin/dmesg | more | grep EXT`

The other command is `/sbin/lspci`, which displays information about all of the PCI busses on the server and the cards connected to them. It operates the same way the `/bin/dmesg` does, here is command syntax that saves the output and prints it to the command shell:

```
/sbin/lspci > pci_info.txt; more pci_info.txt
```

Once you have collected and reviewed the hardware information its time to unpack the kernel source tree. Now, the location of the kernel source distribution needs to be in a specific location `/usr/src/kernel`. Now since is fair to say that more the one kernel distribution could be on a system. The best approach is to untar the kernel distribution, move it to the `/usr/src` directory and then create a symbolic link from the distribution to `/usr/src/linux`:

```
tar xvfz linux-2.4.31.tar.gz
mv linux-2.4.31 /usr/src
cd /usr/src
ln -s /usr/src/linux-2.4.31 /usr/src/linux
cd /usr/src/linux
```

Once you are in the top level of the kernel source tree, the first command you want to run is:

Building A WLAN Proxy Server Part 2

Michael J. Martin

`make mrproper`

This cleans the source tree and purges any configuration files that exist. Now on a new source tree, there will be none of this, but you may want to return the source tree to its original state and now you know how. There are four ways to build the kernel configuration:

`make config`

`make oldconfig`

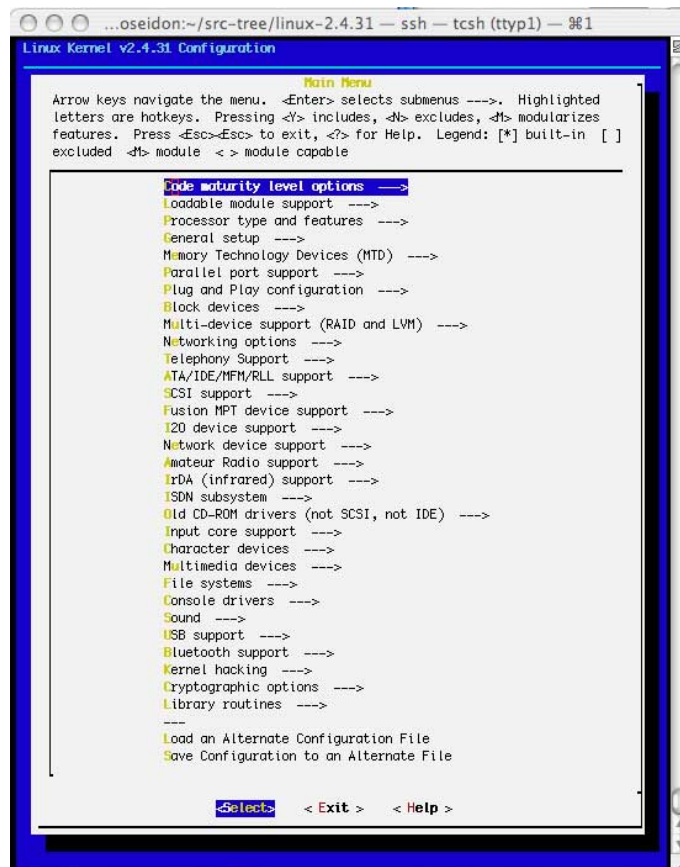
`make menuconfig`

`make xconfig`

The `make config` option is the most tedious; it is a shell script that asks configuration questions sequentially. Simple to use, unless you make a mistake, then you need to start all over again. The `make oldconfig` option reads an existing `.config` file (located in the top of the kernel source tree) and uses that as a basis to build a new kernel config file. What is nice about this option is that if there are any inconsistencies between the old config and new kernel options you will be prompted to resolve them. This is a great way to start building a config file. Once the basics have been "filled" in you can use `make menuconfig`, an ncurses shell menu to tweak the config. Or, if you have XWindows (which you don't) installed you can use the `make xconfig` option to get a GUI menu interface. Since this a new kernel, use the `make menuconfig` option.

`make menuconfig`

Selecting which options to build is largely dependent on the type of hardware your server is comprised of. Here is the Main Menu for the 2.4.31 kernel distribution:



Building A WLAN Proxy Server Part 2

Michael J. Martin

If you have a basic Pentium/AMD server with IDE and built-in graphics you can go with the default settings for the following:

- Loadable Module Support: Enabled by default, provides the ability to load drivers and kernel level services. The `/etc/modules.conf` file defines which modules are loaded at boot. The modules themselves are located in `/lib/modules/`.
- Processor type and features: Specifies which processor the kernel is built for. Pentium-III/Celeron with Symmetric multi-processing support is defined by default. To verify which CPU processor your server is running. Check `/proc/cpuinfo`, using the command:

```
cat /proc/cpuinfo
```

- Block Devices: Defines options for legacy Hard Disk support, certain types of Hard Drive arrays, RAM Disk and floppy support.
- Multi-device support (RAID and LVM) : Defines support for RAID support.
- ATA/IDE/MFM/RLL support: Defines options pertaining to IDE/ATA support, IDE-ATA-2 and IDE/ATAPI support is enabled by default.
- SCSI support: Defines options for SCSI support. If your hard disk, CD ROM, etc is SCSI you will need to build in support for these devices along with the SCSI controller into the kernel.
- File Systems: Defines support for file systems quotas, Ext2, Ext3, DOS, NFS, etc. The Ext3 file system is used by the Red Hat distribution, we need to build in support for the Ext3 file in the kernel:

```
File systems ---->
```

```
 [*] Quota support
 [*] Ext3 journaling file system support
 [*] Microsoft Joliet CDRom extensions
```

- Networking Options: Defines kernel supported network options, such as IP firewalling, 802.1Q and 802.1d support. This option is the focal point for the type of server we are building. To ensure speed and performance the following options are built into the kernel:

```
Networking options ---->
```

```
<*> Packet socket
```

```
 [*] Network packet filtering (replaces ipchains)
 [*] Socket Filtering
 [*] IP: advanced router
     [*] IP: policy routing
     [*] IP: use TOS value as routing key
     [*] IP: verbose route monitoring
 [*] IP: tunneling
 [*] IP: GRE tunnels over IP
 [*] IP: broadcast GRE over IP (NEW)
     IP: Netfilter Configuration ---->
     [*] Connection tracking (required for masq/NAT) (NEW)
     [*] IP tables support (required for filtering/masq/NAT) (NEW)
```

Building A WLAN Proxy Server Part 2

Michael J. Martin

```
[*] MAC address match support
[*] Packet type match support
[*] Multiple port match support
[*] tcpmss match support
[*] Connection state match support
[*] Connection tracking match support
[*] Packet filtering
[*] Full NAT
    [*] MASQUERADE target support
    [*] REDIRECT target support
[*] ARP tables support
    [*] ARP packet filtering
[*] 802.1Q VLAN Support
[*] 802.1d Ethernet Bridging
```

Networking Device Support: Defines the kernel support for network device drivers. This option is particularly critical because without the proper driver support the kernel will not recognize the servers network interfaces. The output from the `/bin/dmesg | more | grep eth` command will provide you with information you need to select the correct drivers to build into the kernel.

Once the kernel build options have been defined. The last step before building is to assign a unique kernel name. This is done by setting the name in EXTRAVERSION field of the default Makefile located in the root of the kernel distribution tree.

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 31
EXTRAVERSION = -SPS-V2
```

```
KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

There are two reasons for customizing the kernel name. First, it makes it easy identify the kernel. Second, the full kernel name is used to identify the kernel modules. Using custom names for each kernel version build from the same core kernel source makes it possible to have different module trees for each variation. Without the custom variable, each new install with overwrites the modules of the previous build.

With the kernel build customizations complete, the build process is a four-step process:

- make depend – builds any supporting components needed by the kernel.
- make bzImage – builds a compressed version of the kernel file.
- make modules – builds any loadable kernel modules defined as part of the kernel configuration.
- make install modules – builds the module directory in `/lib/modules` and copies the modules over.

Installing the new kernel involves a few steps. First, the new compressed kernel image file, `usr/src/linux/arch/i386/boot/bzImage` needs to be installed in the `/boot` directory, renamed in the following format:

```
vmlinuz.version.patchlevel.sublevel.extravesrion.
cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-2.4.31-SPS-V2
```

Building A WLAN Proxy Server Part 2

Michael J. Martin

Once the new kernel file is installed, the next step is to copy over the System.map file. Located in the root of the kernel source tree, into the /boot directory using the same naming format used for the kernel image.

```
cp /usr/src/linux/System.map /boot/System.map-2.4.31-SPS-V2
```

Then new symbolic links for the /boot/System.map and /boot/vmlinuz point to the newly installed System.map and kernel files.

```
rm -f /boot/System.map
ln -s /boot/System.map-2.4.31-SPS-V2 /boot/System.map
ln -s /boot/vmlinuz-2.4.31-SPS-V2 /boot/vmlinuz
```

The last step is to update the boot loader configuration and add our new kernel into the boot loader menu. We configured our server to use the GRUB boot loader. To add our kernel, we need to edit the boot loader configuration file /etc/grub.conf. The file will have the configuration for the kernel we booted the system up with.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#           all kernel and initrd paths are relative to /boot/, eg.
#           root (hd0,0)
#           kernel /vmlinuz-version ro root=/dev/hda2
#           initrd /initrd-version.img
#boot=/dev/hda
default=0
timeout=10
splashimage=(hd0,0)/grub/splash.xpm.gz
title Red Hat Enterprise Linux WS (2.4.21-4.EL)
    root (hd0,0)
    kernel /vmlinuz-2.4.21-4.EL ro root=LABEL=/
    initrd /initrd-2.4.21-4.EL.img
```

Now we need install our kernel boot information above the default configuration so, the server will boot the new kernel by default when the system is rebooted. This is easy to do, by inserting our configuration information, below the splashimage= definition. We can use the current boot configuration as a template and just change the kernel definition and provide a new boot loader menu title:

```
title WLAN Proxy Server Image (2.4.31)
    root (hd0,0)
    kernel /vmlinuz-2.4.31 ro root= LABEL=
```

The syntax of GRUB is a little confusing. For starters kernel definition assumes /boot as the root of the directory tree. So in the example above the kernel file location is an actually /boot/vmlinuz-2.4.31. When defining drives and partitions, GRUB counts drive controllers and drive partitions starting with 0, not 1. So above the actual location of the root partition is on the first controller, first partition. And, were done! Our kernel is built and installed -- now all we need to do is reboot.

Building A WLAN Proxy Server Part 2

Michael J. Martin

If all went as planned you should be running on a freshly minted kernel and are now ready to start the next phase of the server build, configuring networking and building and configuring ISC DHCP, BIND (DNS), SQUID (HTTP Proxy) and Web Proxy Automatic Discovery (WPAD) service. Stay tuned...