Building a Linux Super Kernel for Data Forensics

(revisited)

by Thomas Rude, CISSP

January 2003

Prologue

'Why revisited?' you ask? I originally published this paper in March of 2002. Since then a number of factors have inspired me to do a re-write, and the result is right here, what you are reading. My goal for this version is to clean up the original, simplifying the language and making it more 'newbie friendly'. I am hoping that one result from this version is that I will receive no more E-mails loaded with questions! The “proof is in the pudding” as the saying goes. Did I hit my mark with this rewrite? I would appreciate any feedback. And yes, feel free to send me your questions if you have them.

Before I move into the down and dirty kernel talk I want to take a moment to set the stage. This paper is geared towards Forensic Examiners, whether they be Law Enforcement or Corporate. If you can type faster than you speak and you believe a mouse is something that should be running through a field or taking a siesta in a mouse trap and you understand Magic SysRq key, then consider yourself a Linux guru. This paper is not for you – you guru! Turn around, do not pass go, do not collect $200, go home. No soup for you! But if you come from a point and click world (known in the *nix world as 'move and drool'), say a Win32 environment, and you are interested in customizing the way your Linux system operates, then this paper is for you – read on!

What Is The Kernel?

The Linux Kernel is the brains of the operating system environment. It is a single, compressed file (*vmlinuz-kernel*-XXX) that gets loaded into memory during the boot process. The Linux Kernel then loads modules that are required for the system and mounts the root partition read-only. The kernel is responsible for recognizing and configuring RAM, network connectivity, and configuring storage devices, processor(s), and other hardware.

The Linux Kernel may either be modular or monolithic in design. (More on this later.) Your Linux system may have any number of kernels from which you can choose from during boot. Well, okay, you are limited to the size of your '/boot' partition! But the fact remains clear – you can have one or more kernels to select from during the boot process.

So, is there a benefit to being able to choose a kernel during boot? Absolutely! Each kernel may be customized to a certain environment or for certain hardware, etc. Perhaps one kernel is geared towards network connectivity, including wireless and Bluetooth technologies. A second kernel is geared towards maximum security and as a result, is monolithic in design. Ultimately what is nice is that with Linux you get choice. You get to decide how your environment behaves. And you get to customize that environment if so inclined.

Kernel Terminology

There can be two kernel designs as already mentioned – monolithic or modular. Most users will opt for a modular kernel. However it is important to know that you can elect for either design, and where one design may be more appropriate over the other.

A modular kernel is just that – modular. With a modular kernel modules may be loaded or unloaded as necessary. A modular kernel is usually smaller in size because more options are set as modules and not compiled into the kernel itself. Modules may then be dynamically loaded as they are needed. For example, typically Firewire (1394) support is compiled

as modules that may be loaded as they are needed (when a 1394 device is attached to the system). When no 1394 devices are attached to the system the modules are not loaded as they are not required for operation. However, if we attach a 1394 device we then issue either the 'insmod' or 'modprobe' command to load the necessary modules so that we may use our 1394 device. Note that it may not be necessary to manually load the modules as the operating system itself may handle the loading or unloading of the required module(s).

A monolithic kernel is a kernel wherein all options are compiled directly into the kernel, there are no options selected as modules. This design may result in a larger file size because all options are built into the kernel file itself.

What may be a reason why we would select one design over the other? Being a security freak I can say that my main reason for building a monolithic kernel is out of a desire for a secure environment. Many Linux rootkits are actually kernel modules. If we remove the ability to load modules then we reduce the risk associated with them. Another example may be a simple 486 machine that we use as our firewall gateway. We can actually recompile our kernel so that it will fit on a floppy diskette along with other necessary files and utilities and the end result is a Linux operating system that operates off a floppy diskette. Pop this diskette into the 486 box and we now have a firewall gateway residing on what many folks would consider an outdated PC.

However, for data forensics wherein we may be adding and removing various hardware routinely a modular kernel design is more sensible and that is what we will be making in this paper.

Warning!

Danger! Danger! Take heed – recompiling your kernel may not only result in a inoperable system but may be hazardous to your health and mental stability! One wrong option or configuration and you may find yourself unable to reboot, in a state of kernel panic, or other similar situation. As with anything, please practice on a non-critical system until you have completed the process successfully a few times.

It is also important to note that if you are running Red Hat Linux and you recompile your kernel you have just null and voided your support! Red Hat will *not* support a recompiled kernel. As for the other Linux distributions I do not know how they will treat a recompiled kernel. Best to check their policy before messing with your kernel!

The Red Hat -ac Kernel vs. Vanilla Kernel

There are two branches of the kernel tree; the generic (vanilla) kernel and the Red Hat *ac* kernel. There are noteworthy differences between the two kernels, and I recommend checking out both kernels and comparing them. Perhaps you will find that for your environment one kernel will be more advantageous over the other. Of course you could install both kernels on your system and select between them during boot!

In the past I have chosen to use the vanilla kernel for one reason – patching. There have been times wherein I have needed to apply a patch to the kernel. For example, a file system driver patch such as HFS+ or the ACPI patch. When you patch the kernel there are two results – HUNK SUCCEEDED or HUNK FAILED. With the Red Hat kernel I would get more hunks that failed than with the vanilla kernel. Not being a coder I had a difficult time trying to fix the patches to work with the Red Hat

kernel. Therefore I opted to use the vanilla kernel because hunks did not fail, or did not fail as often as with the Red Hat kernel. Most patches in the wild have been written against the vanilla kernel. Therefore it only makes sense that these patches apply cleanly.

However, as with most things in life there is a flip side to this coin. From my tenure at Red Hat I learned first hand of the rigorous testing and quality control that goes into the *ac* kernel. There are substantial benefits to the Red Hat kernel, and with the majority of corporations running Red Hat Linux wherein stability and quality is mission critical I believe these benefits are evident. Obviously this is not to imply that the vanilla kernel lacks testing or quality control. However, there are big backers with big money using the Red Hat kernel. They can pay for more testing and more control. Got it?

Note also that when installing a Red Hat kernel you are installing a custom kernel built by the Red Hat kernel team. During the installation the 'initrd' image file is automatically created so that you do not have to manually do it. Further, the boot loader configuration file will also be updated to reflect the newly installed kernel (if using GRUB or LILO).

The vanilla kernel may be found at http://www.kernel.org. The Red Hat kernel may be found at http://www.redhat.com.

Kernel Packages

With Red Hat Linux 8.0 there are numerous kernel packages that may be installed, including;

```
kernel-2.4.18-19.8.0 = kernel
kernel-doc-2.4.18-14 = kernel documentation
kernel-pcmcia-cs-3.1.31-9 = PCMCIA card services for laptops
kernel-source-2.4.18-19.8.0 = kernel source
kernel-smp-2.4.18-19.8.0 = multiprocessor kernel package
kernel-utils-2.4-8.13 = kernel utilities package
kernel-bigmem-2.4.18-14 = kernel package for more than 4GB RAM
```

Note that not all these packages will be installed on your system, nor are they all required. The 'kernel-source' package is required for recompiling or kernel development work. The 'kernel-bigmem' package is required for machines with more than 4GB of RAM. The 'kernel-utils' package contains a set of tools that may be used to control both the kernel and hardware.

To find currently installed kernel packages issue;

```
rpm -qa | grep kernel
```

Alternatively, we can issue;

```
cat /var/log/rpmpkgs | grep kernel
```

The first command above will take a bit longer to complete as the system is queried for all installed packages and that output is piped to the 'grep' command for instances of 'kernel'. The second command simply parses a file that is dynamically created at each boot for instances of 'kernel' and therefore is faster in returning results. But make note, this file is created during boot so any package changes since then will not be reflected until it gets updated!

Installing versus Upgrading the Kernel

As with all packages in RPM (Red Hat Package Manager) format we can either install or upgrade. If we opt to install a package we do just that – we install this new package. If we opt to upgrade a package we are doing just that – upgrading the old version of the package to this new one.

With regards to the Linux Kernel I recommend *installing* new kernels over upgrading. By installing the new kernel we will maintain the currently running kernel. Therefore, upon reboot, we will have a choice between the new kernel and the old kernel. If this new kernel for any reason causes our system not to boot we can reboot and select the old kernel that we know works. From there we can work on correcting the new kernel to get it working. Make sense?

If you upgrade your kernel that's it! All you have is that new kernel, and upon reboot if it panics, you only have yourself to blame! Be smart and install, do not upgrade the actual kernel!

However, for kernel documentation and kernel sources upgrading may be the better choice. In fact, we may not have an option as attempting to install new kernel source alongside existing kernel source will cause a conflict.

For this paper we are going to recompile our currently loaded kernel. However it is imperative you learn how to use the 'rpm' program, especially how to install and upgrade packages.

To upgrade to a new kernel issue;

```
rpm –Uvh kernel-XXX.rpm
```

Wherein 'kernel-XXX.rpm' is the new kernel you want to upgrade to. If this completes successfully upon reboot your system will load this new kernel.

To install a new kernel issue;

```
rpm –ivh kernel-XXX.rpm
```

Wherein 'kernel-XXX.rpm' is the new kernel you want to install alongside your currently loaded kernel. Upon reboot your system will have this new kernel as an option to boot.

Both commands above assume you have the kernel in RPM package format, not in a compressed archive format such as TAR.GZ. Please review 'man rpm' and learn how to properly use the 'rpm' program. It is a very powerful program and worthy of learning!

Methodology Overview

At a high level what we are going to do is the following;

- make and test an emergency boot diskette
- query the system for required packages
- customize and recompile the kernel

Creating an Emergency Boot Diskette

This step is *critical*! I cannot emphasize how important this step is. I have received more e-mails and phone calls from folks who have *not* created an emergency boot diskette and upon reboot their system hangs and they do not know what to do next! So much headache, hassle, and time can be eliminated by simply taking this step. But please, don't just create the diskette – TEST IT! Yes, after creating it reboot the system from the diskette to ensure it works as intended.

To create the diskette we issue;

```
mkbootdisk --device /dev/fd0 2.4.18-19.8.0
```

We can see that we specify our floppy device (in this example '/dev/fd0') followed by our current kernel (in this example '2.4.18-19.8.0'). If we are unsure of our current kernel version we can issue the following command;

```
uname -a
```

Sample output from the above command is;

```
'Linux crazytrain 2.4.18-19.8.0 #2 Sat Dec 28 01:38:29 EST 2002'
```

After creating the diskette we reboot our system and boot from the floppy. If we are successful in booting from the floppy we can continue on, but if not, make sure to try again with this step until it works!

Querying for Packages

Since we want to recompile our current kernel we must make sure we have the 'kernel-source' package installed. If it is not installed we must install it before we can recompile. To find out we issue;

```
rpm -qa | grep kernel
```

Hopefully one output will be;

```
'kernel-source-2.4.18-19.8.0'
```

If we have the source package we are golden and can move forward to the next step. If we do not have this package we can grab it from the Red Hat site or the Red Hat Linux 8.0 installation media and install it.

NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE

Yes, please note the following; for this paper actual versions are listed merely for *examples*. These version numbers are *not* meant to be taken literally. I received many e-mails based on the first edition of this paper from folks who were trying to find the specific versions I used for examples! Once again, I list kernel versions here for the sole purpose of examples. You can substitute *any kernel version* for yourself. The version number is not important. I am trying to stress the methodology and options, not the kernel version.

So, for example, if you find kernel 2.4.18-60 and you are currently running 2.4.18-19.8.0, you can either upgrade or install this new kernel and then customize it following the steps in this paper even though I do not reference 2.4.18-60 in this paper. Make sense?

Starting the Customization Process

Please make a note that for this paper we are going to recompile our currently loaded kernel. We want to have the X Window System running so that we can select our options in a GUI environment. How you start X is up to you. Further, we want to have a terminal open so that we can issue commands from the appropriate directory at the prompt.

In order to recompile the kernel we must be in the kernel source directory. The Red Hat Kernel source directory is '/usr/src/linux-2.4' so we must be in that directory to start the process.

```
cd /usr/src/linux-2.4
```

This changes our working directory to the required directory. We can see a number of files, including directories, within this directory;

```
ls -la
```

Before we set our options there is one file we want to review, the 'Makefile' file. Let us issue the following command to view the contents of this file;

```
more Makefile
```

What we are concerned with in this file is the line 'EXTRAVERSION ='. The first couple of lines should make sense to us as we place them together to reveal something like '2.4.18'. The 'EXTRAVERSION' line is the key, though. We edit this line to give our soon to be created new kernel a special name, separating it from all the other kernels on our system.

We must remember what we edit this line to read, though, because we will use this custom name throughout the customization and rebuilding process. For our example I have edited the line such that the 'Makefile' reads;

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 18
EXTRAVERSION = -19.8.0forensic
```

When we are finished with the entire process our new kernel will be the '2.4.18-19.8.0forensic' kernel.

As for how you edit the ‘Makefile’ file the choice is yours. My personal recommendation is to use the ‘Vi’ editor. My reason is simple enough – Vi is installed on almost every UNIX and Linux system. So, if you learn it you will be able to edit files on any system with Vi installed. Please remember that many *nix systems will not have the X Window System installed and so you will not have a GUI environment available. You must learn a terminal (or shell) editor. ‘Pico’ is another shell editor, and it is part of the PINE E-mail package. GUI-based editors include ‘gedit’, ‘kedit’, and ‘nedit’.

One last point I must make clear before moving on. We are going to customize the kernel that resides in the ‘/usr/src/linux-2.4’ directory. You should notice that this is actually a link to the ‘real’ kernel directory. However, if your running kernel is not this kernel you must either reboot and select this kernel or you must change your directory to that of the running kernel. The lead will always be ‘/usr/src’. After that, it depends on which kernel you want to customize.

So now that we have edited our 'Makefile' there is one more critical

step before customizing. If there are any '.config' files in this directory that we want to keep we must copy them to another directory. Again, if you have a '.config' file here that you want to keep copy it to another directory before continuing on with the process.

Hello, Meet Mr Proper

Above I briefly touched on that ‘.config’ files will be removed from our current working directory at the next step (this step). If you want to copy that ‘.config’ file now is the time to do so. For example, to copy it up one level we issue;

```
cp –v .config ../
```

This will copy (‘cp’) the ‘.config’ file up one directory (../) showing us what is happening (v=verbose). So the ‘.config’ file will be copied from ‘/usr/src/linux-2.4’ up one level to ‘/usr/src’.

To remove all configuration files from the kernel source tree we issue;

```
make mrproper
```

This command will remove those aforementioned '.config' files from this directory. (Which is why we copied them to another directory prior to issuing this command in the step above!)

Of course after issuing this command we can copy back a '.config' file into this directory if we desire. The '.config' file is the basis for building a new kernel, it is an ASCII text file that sets all the parameters for the new kernel, which options are turned on and which are turned off, which are compiled into the kernel, and which are compiled as modules. So if you have a custom kernel you like archive off the '.config' file for it somewhere and you can always use it as a basis for building a new kernel!

Dueling Configuration Options

The next step is to actually choose our options. There are a number of choices from which we can choose, including;

make xconfig = to use X Window System GUI environment
make menuconfig = to use a ncurses menu driven shell environment
make oldconfig = to create a '.config' file that shipped with that kernel
make config = to use ncurses menu driven shell environment. NOTE: there is no going backwards here, only forwards, so be careful!

I recommend getting familiar with the first three methods. Xconfig is nice if you have a GUI environment. It is nice and pretty, easy on the eyes. However, there may be a time wherein X11 is not available so getting used to the shell method via menuconfig is a smart thing to do (just as choosing a text installation method for the OS). And lastly, if your options don't result in a kernel that works for your system perhaps choosing the 'make config' method will help because it tends to include support for most everything to get a bootable and working system.

For this paper I am using the 'make xconfig' method so I issue;

```
make xconfig
```

If everything works as it should we will be presented with the image shown below;

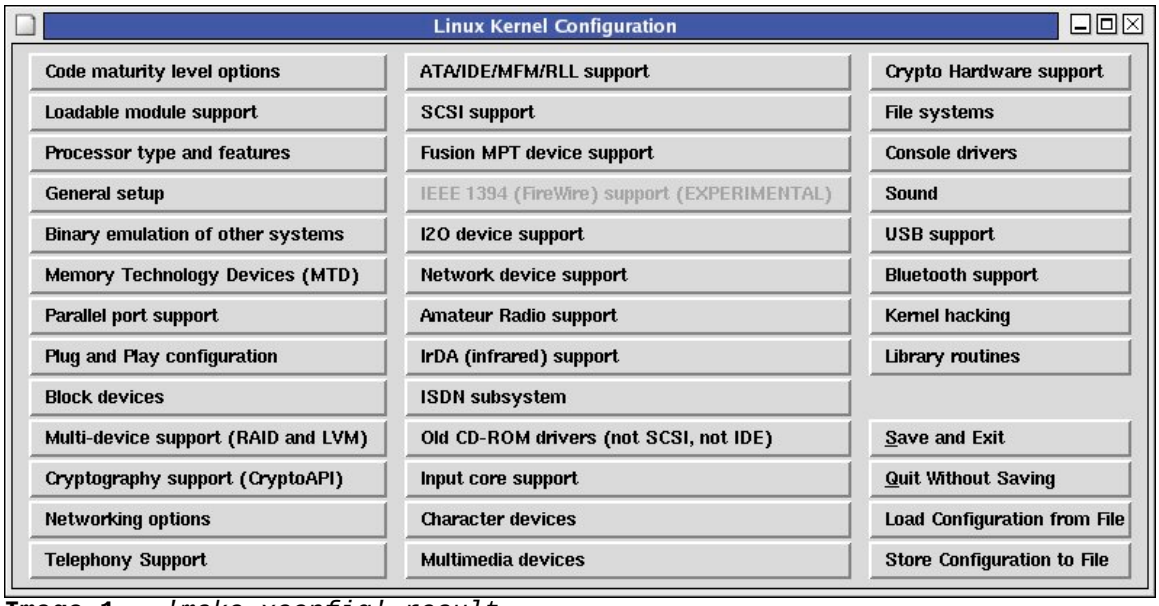

Image 1 *'make xconfig' result*

This is the menu for setting our options. There are a couple of important points to make note of here. One, notice that there may be one or more categories that are dulled out or unavailable as the current '.config' file is written. In *Image 1* above we see that 'IEEE 1394 (firewire) support' is dulled out. Two, there is a method to the madness of selecting options. Just as in reading an English book, we move from top to bottom, left to right. By doing so following selections may become available (or unavailable) based upon earlier selection choices.

CODE MATURITY LEVEL OPTIONS

By clicking on this category we are presented with the window below;

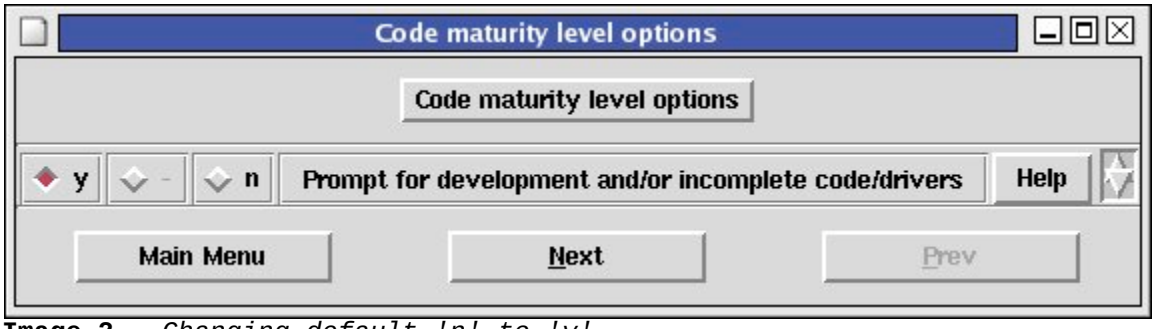

Image 2 *Changing default 'n' to 'y'*

We want to change the default selection of 'n' to 'y' here. In doing so we are saying 'Yes, we are gluttons for punishment! Give us experimental drivers and options, and we will take full responsibility for any beatings that result!' The second we change this to 'y' the previously dulled out 'IEEE 1394 (firewire)' category will become active and available to us, as will a number of other selections within other categories.

I suppose now would be a good time to mention the option choices! They really are self explanatory. 'y' means yes, compile support into the kernel for this selection. 'n' means no, do not compile in support. And 'm' means yes, I want support for this selection, but I want modular support, do not compile support directly into the kernel.

LOADABLE MODULE SUPPORT

Our next area is LOADABLE MODULE SUPPORT. There are three sections here, and I recommend setting the options as shown in *Image 3* below;

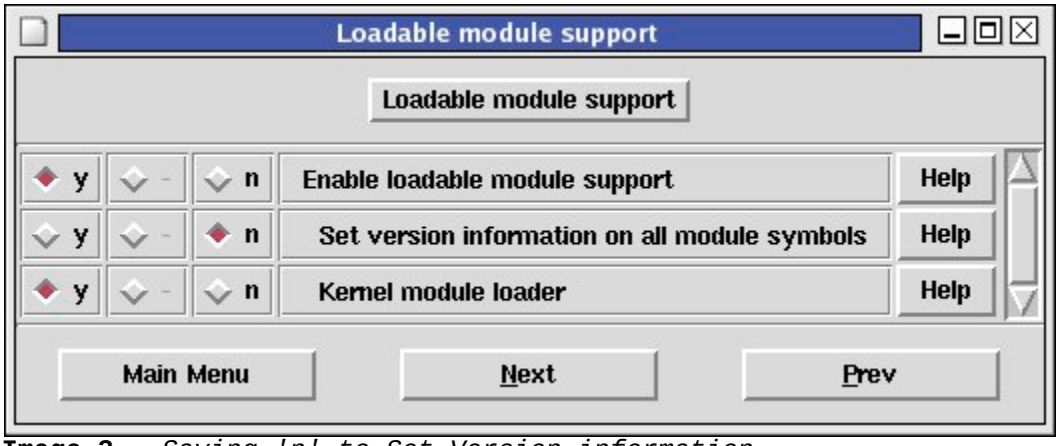

Image 3 *Saying 'n' to Set Version information*

We are making a modular kernel, so we select 'y' to 'Enable loadable module support'. Skipping ahead one we also say 'y' to 'Kernel module loader' and by doing so the kernel will attempt to automatically load modules via 'kmod' (Kernel Module Loader) so that you don't have to. Notice that for the second option though I have selected 'n' to 'Set version information on all module symbols'. If you select 'y' here then you should be able to use modules that were compiled against a different kernel for this kernel. However, my personal experience dictates that I select 'n' and build modules for each specific kernel I build.

And now would be a great time to mention another good thing to do. I strongly recommend grabbing a beverage of your choice, sitting back, and clicking on each of the 'Help' buttons. Not all selections will have a help file available for them, but most will. That's not to say every help file is really helpful. In fact some are quite cryptic and reserved for those living beneath mountains in bunkers! But again, you may learn something quite cool by taking the time to read the help file so just do it!

I will continue on with each of the major categories as they are shown in *Image 1* here on out. To reduce the size of this file I will not include too many images.

PROCESSOR TYPE AND FEATURES

Nothing to cover here for our purposes.

GENERAL SETUP

In GENERAL SETUP category there are a few items to look at. First, I

would recommend selecting 'y' for 'Support for hot-pluggable devices'. Just as the help file says, if you are using a laptop this is a must for swapping PCMCIA cards around! And if you are using a laptop then make sure you click on the PCMCIA/CARDBUS SUPPORT and select the appropriate support for your laptop!

BINARY EMULATION OF OTHER SYSTEMS

The next area I would like to mention briefly is the next major category, BINARY EMULATION OF OTHER SYSTEMS. We can see that if we enable support either via 'y' or 'm' then we can emulate a number of other operating system environments, including; UnixWare, Solaris, SCO, etc. I personally have no use for this selection, but perhaps you do!

MEMORY TECHNOLOGY DEVICES (MTD)

Why might this category by of use to us? Well, if you have to work with RAM chips and/or embedded devices you might find the support you need here in this category.

PARALLEL PORT SUPPORT

If you plan on using your parallel port then look in this category. For example, if you use one of those Backpack CDRW that connect via parallel port then you want to enable support here. I typically opt for modular parallel port support.

PLUG AND PLAY CONFIGURATION

Self explanatory, and I choose;
'y' Plug and Play support
'y' ISA Plug and Play support
'n' PNPBIOS support (EXPERIMENTAL)
As noted, the Plug and Pray BIOS support is experimental in the Linux world. And as such, mileage may vary! Also, since my systems use ACPI I do not opt for PNPBIOS support.

BLOCK DEVICES

Here's a major focus area for forensic examiners! This is where you can select support for specific hardware you have, including hard disks, floppy drives, and even CD writers. Both specific and generic drivers are located here. Read through the choices carefully and find the support you need.

Aside from these devices there are two other very important selections in this category we are concerned with, the loopback driver and the initrd RAM disk.

The loopback device is so important to data forensics, so we definitely want to include support for it. The choice then becomes whether you want the support compiled into the kernel or as a loadable module. I personally choose to compile the support directly into the kernel. If you do this you must remember that by default the Linux operating system allows only 8 loopback devices to be used. You can set more, but you have to tell the kernel this. If you choose support as a module you can pass the parameter when loading the module. If you compile support directly into the kernel you must either plan ahead and pass the option during boot or reboot your system and pass the option. If you are

unfamiliar with what I am speaking of here with regards to passing an option here is an example (with the modular support);

```
modprobe -d loop max_loop=50
```

This will load the loopback module (loop.o) and we see I pass the parameter 'max_loop=50' to the module so that I have access to 50 loop devices instead of the normal 8.

The second selection we are concerned with in this category is the initrd RAM disk. This file is required if you are using the ext3 filesystem or a SCSI controller for your hard drive. This initial RAM disk permits a modular kernel to access and load required modules that it normally would not have access to. The file is located within the '/boot' partition and named *initrd-X.X.X.img* where 'X.X.X.'.

Image 4 below shows these selections and their support choices;

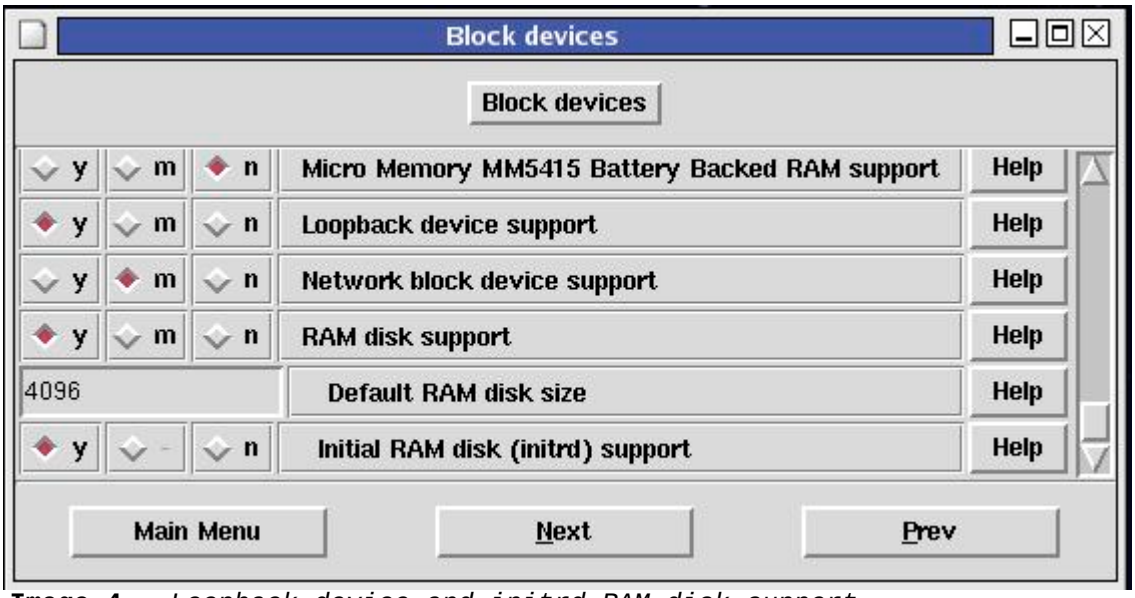

Image 4 *Loopback device and initrd RAM disk support*

MULTI-DEVICE SUPPORT (RAID AND LVM)

If we want RAID and / or logical volume management support this is the category to select the options for us. We all know what RAID is (or should!), but what is LVM? Or, perhaps even better, what can LVM do for us forensic examiners?

The LVM driver will allow us to take several dd created image files, mount them using the loop device, and then view them as one single device. Kind of cool, huh? That volume group of dd image files is like a virtual disk, and as such, we can access it as such.

CRYPTOGRAPHY SUPPORT (CRYPTOAPI)

Making its first appearance in the kernel (without having to patch) is crypto support. Nothing much here for our purposes, but do check out the 'Loop Crypto Support' when you have time. Allows you to create an encrypted loop filesystem. Makes recovering information a bit difficult!

NETWORKING OPTIONS

Obviously the category where networking selections reside. Two sections here that may be of importance to examiners (especially those who do live network analysis) include the IPX and AppleTalk sections. If you need to drop your attack box in an IPX or AppleTalk network this is where you select what support you want for your kernel to have for each of these network protocols.

TELEPHONY SUPPORT

Nothing here that I want to speak of for data forensics.

ATA/IDE/MFM/RLL SUPPORT

An important section to cover for us all. Includes options for IDE/ATA specs, IDE support, CDROM support, floppy support, and tape support. Also very important in this section is the 'SCSI Emulation support' selection – choose 'm'!

SCSI SUPPORT

This category is everything SCSI of course! SCSI disk, SCSI tape, SCSI logging, and SCSI low level drivers. Review this category and select accordingly. Typically I elect;

'm' SCSI support
'm' SCSI disk support (to access 1394 firewire disks)
'm' SCSI tape support
'm' SCSI CD-ROM support
'y' Enable vendor-specific extensions (for SCSI CDROM)
'8' Maximum number of CDROM devices
'm' SCSI generic support
'y' SCSI logging facility

By then clicking on the low-level drivers section you can select the drivers you require, such as those by Acard, Adaptec, Compaq, and Intel. Clicking on the PCMCIA section allows you to include support for those PCMCIA SCSI adapter cards by Adaptec, Future Domain, etc.

IEEE 1394 (FIREWIRE) SUPPORT

My favorite protocol for dumping data, 1394. I am a firewire zealot, if nothing else because it transfers data from one device to another without chewing up my CPU! 'Nuff said!

For 1394 support I opt for the following;

'm' IEEE 1394 support (this is the generic support)
'm' OHCI1394 support (for OHCI driver)
'm' sbp2 support (needed for accessing hard disks)
'm' raw IEEE1394 support (required for data transfers)

Note that if your 1394 device uses the PCILYNX driver then you should select that instead of the OHCI driver support. My personal experience has been great with OHCI devices, and I stick with SIIG wherever I can (both for my laptop and for my desktop). I have never had an issue with their cards and OHCI support. I have had a few issues with those cards using the PCILYNX driver. As a side note, if you use SONY you are on

your own! They are so special they have their own little tweak to 1394 they call it S400. It can be a nightmare to configure with Linux and therefore for all you laptop folks I do not recommend Sony laptops with Linux. Yes, I have the PCG-GRX560, but I am a glutton for punishment. It took me around one week to get everything working on it, which is absolutely uncalled for. But shame on me for not doing my research first and finding the onboard 1394 controller is a Ricoh controller which equals not so good Linux support!

I2O DEVICE SUPPORT

Nothing here to report on from a forensic point of view.

NETWORK DEVICE SUPPORT

This category is where you will choose your network card drivers and the like – Ethernet, token ring, wireless, etc.

AMATEUR RADIO SUPPORT

Again, nothing to report on here for forensics!

IRDA (INFRARED) SUPPORT

Okay, has this died yet or is their a comeback coming? If you want IrDA support then you will find it here. I don't use it – even with silver pants!

ISDN SUBSYSTEM

If you have ISDN here is your support.

OLD CD-ROM DRIVERS (NOT SCSI, NOT IDE)

If you have old hardware lying around perhaps you will need to review the selections here to include support for your old CD-ROM.

INPUT CORE SUPPORT

If you plan on using a USB keyboard or mouse then enable support here for those items.

CHARACTER DEVICES

There are a few things of interest here. If you use the floppy tape device driver (ftape) then here is where you will find it and the selections available for it. Besides ftape you will also find your video card support here. But lastly, and this is important, there is a section 'PCMCIA Character Devices'. If you use a laptop you will want to visit this category and select the support you want. I typically choose the following;

'm' PCMCIA serial device support

Why is this important to the forensic examiner? If you plan on using a PCMCIA compact flash adapter card then this driver may be required so

that you can access that flash like an IDE disk.

MULTIMEDIA DEVICES

Nothing here from a forensic point of view.

CRYPTO HARDWARE SUPPORT

Nothing here from a forensic point of view.

FILE SYSTEMS

Obviously a biggie for forensic examiners! I typically include most read support, but limit 'write' support to only a few drivers. Review the list of file system drivers and include support for what you need. I recommend compiling modular support for most of them because you never know when you will need them!

At the bottom of this category we see there are 'Network File Systems', 'Partition Types', and 'Native Language Support' categories. Peruse each, including support for those desired. Again, for the file systems I include most everything. The same for partition types. Native language support I include those that I use often.

CONSOLE DRIVERS

Nothing here from a forensic standpoint.

SOUND

Nothing here from a forensic standpoint.

USB SUPPORT

Another category of importance for forensic examiners. You will find support here for the different USB drivers, compact flash readers, CD-Writers, digital cameras, and more.

Note the 'USB Serial Converter support' category. This category is where you can find support for Handspring Visors, Palms, Sony Clie, Compaq iPAQ, HP Jornada, and Casio handheld devices. Good stuff here.

BLUETOOTH SUPPORT

To have Bluetooth support this is where you will find it.

KERNEL HACKING

Nothing here from a forensic standpoint.

LIBRARY ROUTINES

Nothing here from a forensic standpoint.

Okay, there we have it. We have gone through and reviewed the categories and hopefully we have selected options beneficial to our system and environment. We click on 'Save and Exit' which will write the '.config' file for us and close the window.

Note that we could have chosen to 'Quit without saving' wherein all changes would be lost. Also note the 'Load Configuration from file' option. If we clicked on this when we first started we could have pointed to a '.config' file in a target directory we specify.

Setting the Dependencies

Now that we have selected our options we must set the dependencies to ensure that we have everything we need for a working system. Issue;

```
make dep
```

This sets all the dependencies correctly. Remember that we must be in the ‘/usr/src/linux-2.4’ directory when issuing all of our commands! (or, in the kernel directory you are customizing)

Take Out the Trash

Time to prepare the kernel source tree for building the kernel;

```
make clean
```

Kernel Time

Next we build the actual kernel via;

```
make bzImage
```

Please remember that Linux is case sensitive. Upper and lower case counts for everything!

Module Time

Time to make our modules;

```
make modules
```

When this completes the process of making our modules we follow with;

```
make modules_install
```

This installs the modules for our custom kernel to '/lib/modules/kernel-X-X-X' where 'kernel-X-X-X' is the name of the kernel.

Please note that even if you built a monolithic kernel you must issue these two commands as part of the kernel building process!

Installation Time

To install our new kernel on a Red Hat system we issue;

```
make install
```

This copies our new kernel and the associated files (System.map) to the appropriate directories. Further, 'new-kernel-pkg' is executed which in turn builds the initrd image file and updates the boot loader configuration file to reflect the new entry.

It is important to check to make sure the important files found their way home and the boot loader configuration file was, indeed, updated.

We want to look into the '/boot' directory to make sure there are three new entries reflecting our new kernel. These files are;

```
vmlinuz-kernel-version
System.map-kernel-version
initrd-kernel-version
```

For example, in our example we would then have these three files added;

```
vmlinuz-2.4.18-19.8.0forensic
System.map-2.4.18-19.8.0forensic
initrd-2.4.18-19.8.0forensic.img
```

Remember how I mentioned it is important to remember what we edit the EXTRAVERSION line of the 'Makefile' to be? Here is why!

I use GRUB as my boot loader, and the file to check to make sure the new kernel is included as an option during boot is '/boot/grub/grub.conf'.

If you use LILO as your boot loader then your config file is '/etc/lilo.conf'.

Review

In short, here are the steps to take;
1) create emergency boot diskette
2) edit ‘Makefile’ to customize new kernel name
3) copy any ‘.config’ files to another directory
4) issue ‘make mrproper’
5) issue ‘make xconfig’ or ‘make menuconfig’ or ‘make oldconfig’
6) issue ‘make dep’
7) issue ‘make clean’
8) issue ‘make bzImage’
9) issue ‘make modules’
10)issue ‘make modules_install’
11)issue ‘make install’
12)check boot loader configuration file ‘/etc/lilo.conf’ or ‘/boot/grub/grub.conf’
13)check ‘/boot’ for presence of new files;
initrd-kernel-XXX.img
System.map-kernel-XXX
vmlinuz-kernel-XXX

Epilogue

If you have successfully recompiled your kernel and booted your system using it I have achieved my goal! Congrats to you! Enjoy your new kernel.

Customizing your kernel really is not that difficult of a process once you learn the steps and become accustomed to the methodology. And what do you do when good times go bad? You boot back into your old, working kernel and try to fix the problem with the non-working kernel. Yes, I have found myself facing the all too familiar “kernel panic” message wherein my only resort is to hard power off the system. This is typically because ‘initrd’ wasn’t created or I left out some option in the kernel (perhaps a BIOS or PCI option). But getting these error messages is actually helpful as it will force you to troubleshoot and learn how to solve issues.