Secure Network Configuration Management
for Linux based Routers

GIAC Certified UNIX Security Administrator (GCUX)
Practical Assignment Version 3.0 - Option 1

Ronald Young
March 14, 2005
Las Vegas, NV (10/04) Course

Abstract

This paper presents a detailed implementation and operation plan for remote configuration management of a research network infrastructure. GIAC University is currently involved with several large-scale research projects that utilize individually identifiable medical records. Medical records are considered protected data by the U.S. Health Insurance Portability and Accountability Act (HIPAA) and steps must be taken to insure their confidentiality. Until now, the University has had complete control over the computing infrastructure for its projects. Recently, a joint partnership between the university and a private company (ACME Industries) was awarded a federal grant to computerize a collection of historical medical information. This collection will be used for ongoing medical surveillance of individual patients and other industrial health applications. Users of these applications will be located nationwide and will access the data via a private network as well as the Internet.

Data security for previous projects was achieved by a combination of physical and network/host-based security measures. The application systems were located in a remotely monitored secured room with card-key access. Network access was controlled by a traditional set of routers, firewalls, and intrusion detection systems which were also placed in secure access-controlled locations. All network traffic was required to be transferred in an encrypted form using either of the SSH (secure shell) or SSL (secure session layer) protocols. Responsibility for network and data security resided with GIAC University.

Access to the data and application programs by users was managed by issuing unique usernames and passwords to each authorized user. A password is only valid for a limited time, after which it must be changed by the user. There was also a documented standard for selecting a password which is enforced by the password change utility.

With the addition of the ACME Company, this security model must be extended to protect the computer and network equipment located at their offices. Because of limitations in the physical plant of the building housing the ACME offices, the network equipment must be placed in a location that could be possibly accessed by ACME staff and visitors. An added complication is that the network security staff is stationed at the main location instead of the ACME offices.

The rest of this paper consists of the following sections:

- A description of the environment of the current network and application security infrastructure. A detailed description of the hardware, software, and network environment focusing on two specific systems. One functioning as a core network router and the other as a network management system.

- A proposed change to the network infrastructure to help minimize the risk of a compromise of the network resulting from an attack on the core router originating

from the ACME offices.
- An analysis of the existing security issues that the proposed change is intended to address. A discussion of the new security issues introduced by the proposed change as well as steps that can be taken to mitigate them.

- A detailed description of how to install and configure the proposed change.

- A procedure to verify the initial installation.

- A periodic maintenance and audit procedure to verify that the change continues to function in an effective and secure manner.

- Finally, a summary and discussion of areas for possible future research and enhancements are presented.

You will also notice throughout the rest of this document that there are paragraphs that begin with a number followed by (___). These form a detailed checklist that can be used to install and configure the new boot service described. You can place a checkmark or initials inside of the (___) as each step has been completed.

Initial Environment

This section describes the initial hardware, software, and network environment that is in place before the proposed changes are made. The justification for choosing a particular operating system distribution and configuration options is also presented. In addition, we will also describe the security policies that govern physical and network access by authorized users.

Hardware Description

The network infrastructure is made up of multiple commodity (32-bit Intel processor based) systems. The actual configuration of each system depends on its function, load, and exposure to potential threats. For the purposes of this document, we will describe two configurations: a core network router and the network management console used to monitor and control the network. Commodity systems are used instead of specialized equipment like Cisco routers and PIX firewalls because of their significantly lower cost and additional flexibility in configuration. GIAC University prefers to use Dell rack mountable servers because of the wide choice of configurations and the existence of a volume purchasing agreement with Dell.

The typical hardware configuration of core network router is a rack mountable Dell PowerEdge 2650 system. The base system consists of the chassis, dual 2.8 GHz Xeon processor (w/ SMT “hyper threading” capability), a PERC/3 SCSI (a Dell rebadged Adaptec) Ultra 320 RAID controller, two 38GB disk drives, and two on-board 10/100/1000mb/s Ethernet NICs. Additional network interfaces are added as necessary. In this case, the router is one of a pair of routers that form a dedicated gigabit Ethernet (gigE) fiber link. The interface for the fiber connection is an Intel PRO/1000MF NIC (Part #PWLA8490LX). Two additional 10/100/1000mb/s interfaces are made available by adding an Intel PRO/1000MT Dual Port card (Part #PWLA8492MT). It can be argued that this machine configuration is overkill, but having excess capacity allows us to minimize the risk of the network failing because of a denial of service attack that overloads the router. It also allows us to run several internal intrusion detection sensors and other network monitoring software without having to use additional dedicated systems[1]. The slightly increased risk of having the monitoring software on the routers is offset by the increased hardware costs for separate systems.

Intel was chosen as the add-on network cards since their PRO/1000 line of cards utilize a common network driver.[2] Also, using Intel cards allows for a "drop-in" upgrade to 10GbE on our network fiber links since the PRO/10000 cards also use the same driver as the existing PRO/1000 cards.

The hardware configuration for the network management system is compatible with the above. It is a Dell PowerEdge 750 1U rack mountable system. The system consists of a single 2.4 GHz processor, 512MB ram, 40GB SATA disk, and 2 onboard network interfaces.

Software Description

The network infrastructure uses a custom Linux (Gentoo-based)[3] distribution. Why choose Gentoo Linux another Linux or *BSD distribution? The primary reason was two-fold: Gentoo Linux is one of the few distributions that utilize a "build from source" philosophy. The system installer can build the entire operating system (including kernel, compilers, and libraries) from source code[4]. This makes hardening the system much easier and less error-prone because:

- We are sure that the system can be rebuilt from known sources and patches, since this is how the system was first installed.
- One of the techniques used in hardening a system is the removal of unused packages. Since Gentoo starts as a minimal system of less than 60 packages, it

---

[1] NOTE: The network monitoring and IDS systems are in addition to a dedicated set of IDS servers used to monitor an internal firewall protecting the main database server.

[2] For the Intel PRO series interfaces, Linux uses the e1000 and OpenBSD uses the em drivers.

[3] Gentoo Linux is a source code based distribution that is very customizable. The project home page is located at http://gentoo.org.

[4] The Gentoo distribution uses the concept of "stages". Stage1 builds everything from scratch, stage2 builds the system from a bootstrapped "semi-compiled" state, and stage3 is similar to a regular Linux distribution in that almost everything in the system is loaded from precompiled binaries. See http://www.gentoo.org/doc/en/handbook-x86.xml?part=1&chap=2 for more information.

is much easier to insure that the system is as small as possible.

- Also since the system is built from source, it is very easy to control the options used to compile and install a package. For example, to include SSL support for a Gentoo system, the admin only needs to add the string “ssl” to an environment variable and “remerge” the system

.

- The Gentoo system management paradigm provides a rich set of command line utilities that allows the system administrator to easily control system services, i.e. “rc-update add sshd default” configures the system to have the SSH daemon automatically during a normal boot.

The Gentoo distribution has excellent installation instructions available on their web site.[5] Gentoo also has a wide selection of online message forums available. Because of their high quality, please refer to them for general installation information. Throughout the remainder of this document, only installation information that is directly relevant to system security will be presented.

Before the freshly installed server is connected to the network, additional steps must be taken to harden it. This additional hardening process varies by the OS, but the results are the same: to help protect the system from threats from hostile people on the Internet or even unintentional damage from authorized users.

Hardening a Gentoo Linux system is easier in the sense that throughout the installation process the installer has control over which software packages and what options they are installed with. This is different from a typical “mainstream” Linux distribution like Red Hat, where the installer is forced to choose from a small list of vendor determined installation configurations (i.e. minimal, desktop, developer, or server). Then according to this option, numerous precompiled packages are installed. To harden this type of system requires the installer to inspect each of the packages loaded on the system and manually remove the dangerous or unnecessary ones. Once these packages have been removed, the remaining ones may still need additional work to securely configure them, or even need to be recompiled from source.

The following example shows the normally running processes for the Gentoo Linux core router after hardening. The number of “processes” shown is somewhat misleading, they are not all true processes. The command names enclosed in brackets (i.e. [migration/0]) are not real processes but instead are “kernel threads”. They are built into the kernel and only show up in the process table for scheduling purposes.

```
PID TTY      STAT   TIME COMMAND
   1 ?        S      0:01 init [3]
   2 ?        S      0:00 [migration/0]
   3 ?        SN     0:00 [ksoftirqd/0]
   4 ?        S      0:00 [migration/1]
```

---

[5] The latest installation document for Gentoo can be found at http://www.gentoo.org/doc/en/handbook/index.xml.

```
   5 ?        SN      0:00 [ksoftirqd/1]
   6 ?        S       0:00 [migration/2]
   7 ?        SN      0:00 [ksoftirqd/   8 ?        S       0:00
[migration/3]
   9 ?        SN      0:00 [ksoftirqd/3]
  10 ?        S<      0:00 [events/0]
  11 ?        S<      0:00 [events/1]
  12 ?        S<      0:00 [events/2]
  13 ?        S<      0:00 [events/3]
  14 ?        S<      0:00 [khelper]
  15 ?        S<      0:00 [kacpid]
  44 ?        S<      0:00 [kblockd/0]
  45 ?        S<      0:00 [kblockd/1]
  46 ?        S<      0:00 [kblockd/2]
  47 ?        S<      0:00 [kblockd/3]
  48 ?        S       0:00 [khubd]
  58 ?        S       0:00 [kirqd]
  59 ?        S       0:00 [pdflush]
  60 ?        S       0:00 [pdflush]
  61 ?        S       0:00 [kswapd0]
  62 ?        S<      0:00 [aio/0]
  63 ?        S<      0:00 [aio/1]
  64 ?        S<      0:00 [aio/2]
  65 ?        S<      0:00 [aio/3]
 650 ?        S       0:00 [kseriod]
 672 ?        S       0:00 [scsi_eh_0]
 673 ?        S       0:00 [aacraid]
 677 ?        S<      0:00 [ata/0]
 678 ?        S<      0:00 [ata/1]
 679 ?        S<      0:00 [ata/2]
 680 ?        S<      0:00 [ata/3]
 691 ?        S       0:00 [khpsbpkt]
 697 ?        S       0:00 [kjournald]
 826 ?        Ss      0:00 /sbin/devfsd /dev
4991 ?        S       0:00 [kjournald]
4992 ?        S       0:00 [kjournald]
4993 ?        S       0:00 [kjournald]
4994 ?        S       0:00 [kjournald]
5705 ?        Ds      1:03 /usr/sbin/syslogd -m 0 -r
5709 ?        Ss      0:03 /usr/sbin/klogd -c 3 -2
5845 ?        Ss      0:00 /usr/sbin/sshd
5863 tty1     Ss+     0:00 /sbin/agetty 38400 tty1 linux
5864 ?        Rs      0:00 sshd: root@pts/0
5870 pts/0    Ss      0:00 -bash
5874 pts/0    R+      0:00 ps agx
```

In order to reliably secure a UNIX system, the security administrator must have a thorough understanding of what system components are necessary for proper operation and their function. Here is a brief description of each of the processes shown above:

- *init* (pid 1): this process is started by the kernel during the system boot process. It begins execution after the kernel has activated essential operating system functions (i.e. memory management, loaded device drivers, etc). Init is the parent process of all other processes running in the system. If it aborts, the system will crash. It is also the first process that executes using a normal user environment (i.e. it can use the normal C runtime library).

- *migration* (pids: 2, 4, 6, and 8): there is a migration thread active for each enabled CPU found in the system. These threads manage the migration of processes between CPUs.
- *Ksoftirqd* (pids: 3, 5, 7, and 9): like the migration threads, there is a ksoftirqd thread for each active CPU. They manage the distribution of hardware interrupt requests for processes that are running on the associated CPU. Kirqd (pid: 58) services the actual hardware interrupts for distribution to the appropriate ksoftirqd instance.
- *events* (pids 10-13) are kernel threads that control the dispatching of operating system signals and events to processes. These events can indicate possible error conditions (i.e. memory access errors SIGSEGV, floating point errors SIGFPE, user or program interrupts and aborts SIGINT, SIGQUIT, and SIGABRT). It can also report interval timer expirations (i.e. waiting time for keyboard input has elapsed).
- *khelper* (pid: 14) is a kernel thread that helps manage the loading and unload of system modules for user mode programs. It replaces the *kerneld* daemons found in earlier Linux versions.
- *kacpid* (pid: 15) manages the Advanced Configuration and Power Interface hardware found on most modern computer systems. ACPI can monitor things like system temperature, battery status, etc. It has limited usefulness for always-on servers and can be removed by rebuilding the kernel.
- *khubd* (pid: 48) manages the plugging and unplugging of USB devices.
- *kblockd* (pids: 44-47) manages the block device buffer cache for each active CPU.
- *pdflush* (pids: 49-60) works with kblockd to write blocks found in the buffer cache to disk.
- *kswapd0* (pid: 61) manages the swapping of processes into and out of memory.
- *aio* (pids: 62-65) controls asynchronous I/O for each CPU. Asynchronous I/O is normally character oriented (i.e. serial devices and network traffic).
- *kseriod* (pid: 650) communicates with the hardware drivers for serial devices.
- *scsi_eh_0* (pid: 672) is the kernel thread that handles error and timeout handling for SCSI devices.
- *aacraid* (pid: 673) is the hardware driver for the system's SCSI RAID controller.
- *ata* (pids: 677-680) controls ATA/IDE devices (i.e. CD-rom and IDE disk drives).
- *khpsbpkt* (pid: 691) handles communication with "high-performance serial bus" devices like IEEE 1394 (Firewire).
- *kjournald* (pids: 697, 4991-4994) manages the journal log file for ext3 based file systems.
- */sbin/devfsd* (pid: 826) is a user mode daemon that is used by the system to manage the Linux Device Filesystem (devfs). Devfs allows for the dynamic creation and removal of node entries for things like USB and Firewire devices
- */usr/sbin/syslogd* and */usr/sbin/klogd* (pids: 5705 and 5709) daemons that allow processes and the kernel to send messages to log files.
- */usr/sbin/sshd* is a controlling daemon that accepts incoming network requests to establish a secure communication method for terminal sessions. For each connection another copy of the daemon is created (pid: 5864). The user of that

terminal session is using the bash shell (pid: 5870) and is executing the "ps" command (pid: 5874).
- */sbin/agetty* (pid: 5863) allows terminals connected to a serial interface to connect the system. It will configure the serial port for communications (i.e. set baud rate, parity, and terminal type) and then prompt for a username. It then executes the login program to prompt for a password and to verify the user information.

In addition to the above processes, management of information stored on magnetic media (hard drivers) has a large impact on system security. Steps must be taken to minimize the possibility of unauthorized access or modification of information stored on the system (both programs and data). UNIX-like systems store information on hard drives using the concepts of files and directories. A directory contains the name of one or more files or other directories. A file may contain data or programs or a link into the operating system that points to a device (i.e. /dev/tty1 refers to the first serial port on the system). Each directory and file has an "owner" and what operations (read, write, execute) are allowed by various groups of users.

A first step to accomplish this is to partition the disk storage into multiple file systems. Each file system holds a specific set of files with their own protection requirements. The following table shows the current file system layout for the core router.

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda5               489992     33380    431312   8% /
/dev/sda6               988212     16428    921584   2% /home
/dev/sda7              3945128   1276488   2468232  35% /usr
/dev/sda9              1976492     51660   1824428   3% /var
/dev/sda8             19694836     34140  18660252   1% /var/tmp
none                    257436         0    257436   0% /dev/shm
```

The root file system (/) contains files and directories that are critical to system startup and operation. Only the system administrator should be able to create, modify, or delete files. The home files system (/home) contains the files and directories for system users. Each user should have their own home directory such that they are the only ones that can access or modify files contained inside of it. Only the system administrator should be able to create new home directories. The /usr file system contains the programs and data files for packages installed on the system, this is where things like compilers, libraries, and utilities are located. Again, only the system administrator should be able to add or modify information in /usr. It should be read/execute only for all other users. System information that needs to be updated is stored on the /var file system, this is where things like mail messages, system information logs, and other similar files are stored. The permissions on these files and directories show prevent normal users from access or modification except under the control of trusted system utilities. An area to hold temporary files for normal users is available in /var/tmp. While any normal users may use this area, files and directories are normally protected so that only the file owner may access or modify them.
The last entry allows the GNU C-library to implement POSIX shared-memory.

In addition to the above, there are several others that are used by the system. The boot (/boot) file system is normally not available when the system is running. It contains the configured operating system kernel and boot loader that is loaded when the system is booted. Only the system administrator should be able to modify (or even read) information contained on /boot.

The remaining file systems are used by user mode programs to communicate with the operating system: /dev, /proc, /sys, /dev/pts, and /proc/bus/usb. /dev is used by the system and /sbin/devfsd to create special file entries to allow users to access hardware devices. The /proc and /sys file systems allows users and programs to query and possibly change parameters used by the operating system kernel. The system uses these special file systems (known as pseudo file systems) used so that the normal protection methods can apply to them.

Physical and Network Environment

Physical access to the network equipment located in the main computer room at GIAC University is strictly controlled. Only the staff members whose job functions require them to be inside of the computer room are permitted access. This access is controlled by the presence of magnetic badge readers. Access by anyone else requires that they by escorted by an authorized staff member whenever they are inside of the controlled area. The room also has an alarm system that monitors all of the doors and windows of the room. Both the badge reader and alarm system are remotely monitored by an external agency (the U.S. Department of Energy).

Access to network and computer resources are controlled by a combination of network and host based protection mechanisms. The network protection consists of a set of firewalls and intrusion detection sensors deployed throughout the network. Traffic passing into or out of the network to the Internet is managed by a gateway (bastion) host. Authorized access to the network and its hosts are controlled by users entering a username and password. Passwords must conform to a minimal security standard: be at least 8 characters in length, contain at least 2 upper case letters, one special character, and 2 numeric digits. They must be changed once every six months, and cannot be "too" similar to previously used passwords.

Every device and host connected on the network must display a warning message whenever a connection occurs. The content of this message is determined by the granting agency for the project. The message for the current project is:

> NOTICE TO USERS
>
> This is a Federal computer system and is the property of the United States government. It is for authorized use only. Users (authorized or unauthorized) have no explicit or implicit expectation of privacy. Any or all users of this system and all files on this system may be intercepted, monitored, recorded, copied, audited, inspected, and disclosed to authorized site, Department of Energy, and law enforcement personnel, as well as authorized officials of other agencies, both domestic and foreign.

By using this system, the user consents to such interception, monitoring, recording, copying, auditing, inspection, and disclosure at the discretion of authorized site or Department of Energy personnel. Unauthorized or improper use of this system may result in administrative disciplinary action and civil and criminal penalties.

By continuing to use this system you indicate your awareness of and consent to these terms and conditions of use.

If you do not agree to the conditions stated in this warning,
LOG OFF IMMEDIATELY

New Software Service

Due to the lack of dedicated network staff located at the ACME offices, it is desirable to manage the network infrastructure from the main site as much as possible. We also want to try and mitigate the risks of the lower physical security of the ACME offices by improving the remote monitoring and configuration capability.[6] Because of the sensitive nature of the data, the default response to network failures and attacks should be to "fail closed" (prevent the network from being able to process data). This is different from the normal "fail-safe" default for more open networks (continue processing data even in light of a denial of service attack). To address this, we will attempt to develop a mechanism to boot a server using an operating system image provided by a remote server while minimizing the security risks. While this study focuses on managing network routers, this method can be used for other types of "appliance" systems like thin clients, kiosks, file servers, and similar applications.

Why develop a new network boot method? Wouldn't existing ones like PXE[7] and etherboot[8] work? While both of these methods allow for a system to boot from the network, they do so in an insecure manner. Both PXE and etherboot rely on information obtained from network servers using the BOOTP, DHCP, and/or TFTP protocols & utilities. These utilities do not normally support client authentication, encrypted data streams, or reliable error recovery. They are all UDP based protocols. Also, the size of the bootstrap image supported by them is typically limited to 16Kb or 32Kb.

The intent of our changes is to try and overcome these restrictions. This will allow the remote management of routers in a secure manner without requiring the presence of network staff on-site. The new method will function as follows:

- A minimal boot image is stored on a bootable read-only media device (i.e. CD-

---

[6] It is important to remember that it is impossible to fully secure a computer system if an attacker can gain physical access to it. The most that we can hope for is to be able to detect and mitigate such attempts.

[7] PXE stands for *Pre-boot Execution Environment*. It is a standard developed by PC based hardware and software vendors to allow PCs to boot over the network.

[8] Etherboot is an open source project that provides ROM images suitable for installation on network interface cards that will allow the PC to boot over the network. See http://etherboot.sourceforge.net for more information.

rom, or USB memory stick). This boot image loaded when the system (our router) is powered-on or reset. It uses the network interface to establish network session to a secured server to download the production operating system and configuration for the router into its memory. The new operating system kernel contained in memory is then started.

- The production operating system will mount a remote file system to store system logs, configure and start the network services, and finally, start a secure shell daemon to allow access to the router by the remote network staff.

Issues introduced by the new boot method

By changing the way that the system boots, several issues with security implications are introduced, among them:

- Unauthorized substitution of the initial boot media. This is the hardest issue to address. Usually, this is issue is handled by improving physical security to the room housing the system and allowing only trusted personnel access. Since this is not a feasible option, the steps that we will take to mitigate this are: store only non-sensitive information on the media, mount the media early in the boot process in order to “lock” the drive, and have the production image verify the media contents as part of its boot process. None of these steps are fool-proof they only serve to help minimize the risk. Other steps that could be taken include using a “disk on a chip” to store the initial boot media or use a custom BIOS[9]. The best method of addressing this issue is to improve physical security and limit personnel access.

- Updating the initial boot media. How do you update the initial boot media when needed? This can be handled by a combination of procedural and technical means. The revised boot image is copied onto new media and delivered to the ACME office manager (either electronically or physically). Once the new media is ready to be installed, the following steps are used: 1) the central site network staff issues the “unlock” command on the drive and ejects the old media; 2) the office manager installs the new media in the drive; 3) the central site network staff mounts (locking the drive) and verifies that the media is valid; 4) finally, the old media is returned to the central site or destroyed.

- Sniffing and/or altering the data stream (man-in-the-middle attacks). Since we are now using a remote system to store production operating system images, the possibility exists of an attacker capturing or altering the data during transmission. We address this by a combination of physical and network methods. A point-to-point fiber optic link is used between the sites. This makes “sniffing” the wire at

---

[9] We could use something like a modified LinuxBIOS (http://www.linuxbios.org) to directly access the remote server. However, this has its own set of problems, not the least of which is supporting the hardware devices on the system.

any point other than the ends more difficult. An additional step is to declare the link “insecure” and require all traffic to be further encrypted using SSL, IPSEC, or similar protocols.

- Exhausting network resources. This issue exists in general for any network service. The fact that we are using the network to load the production image from a remote server and store log files remote increases our exposure to a denial of service. We address this issue by using a “heartbeat” on the network connection to the remote system. A heartbeat is something that is used to determine that a remote system is operational. In our case, the system that houses the images and log files for the remote routers will monitor their network traffic and if an outage is detected, disable the network interface and issue an alert to the network staff. This will result in a complete interruption of service. Since the network outage is total, the sensitive data is protected in accordance to operating procedures and the network staff can investigate and correct the cause. The protocol used for remote access of the images and log files have built-in retry timers to help recover from transmission errors. Each time a retry occurs, a message is written to the system log. By having a program watch the log files and periodically “ping” the application on the remote server, an outage can be detected. Once detected, the script can issue system commands to disable the network link and to alert the network staff.

- Resource exhaustion on the local and remote logging server. Since we are logging information into a fixed size circular buffer, information could be lost if it overflows. Similarly, if the remote logging server’s file system overflows, log information will also be lost. This could be addressed by isolating the logs for each router in file systems separate from the logging system and the other routers. Then, a file system overflow will only affect a single server. Also, the management of the individual log files becomes a simple system administration problem: rotation and analysis of log files.

Implementation

There is no single package that is capable of providing everything that we need in order to secure our routers. However, by using a combination of several open source packages, we can build our own. Our custom boot process will use the following packages and utilities:

- A Linux boot loader, there are several boot loaders available. We will be using the isolinux boot loader which is part of the syslinux package (home page: http://syslinux.zytor.com). The version of isolinux that we will be using for this project is part of the Gentoo distribution (emerge sys-boot/syslinux). *Isolinux* is stored on a CD-rom and called by the hardware BIOS to boot the CD.

- *BUILDROOT* (home page: http://buildroot.uclibc.org) is a set of makefiles and patches that makes it easy to generate a cross-compilation toolchain and root `filesystem for working with Linux on embedded systems. I`t uses the *uClibc and BusyBox* packages. *uClibc* is a small footprint C library (home page: http://www.uclibc.org). *BusyBox* (home page: http://busybox.net) provides small footprint replacements for the standard UNIX utilities like grep, md5sum, init, etc. Our intent is to use *BusyBox* statically linked with *uClibc* to provide the support utilities for our router software distribution[10].

- *Linux Kernel 2.6.8.1* (http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.8.1.tar.gz) is the kernel version that we will be using for our custom router distribution.

- *vtun* (Virtual Tunnel) is actively developed by Maxim Krasnyansky and the rest of the Vtun development team. Its purpose is to allow the easy creation of secure data transport tunnels. The project's home page can be found at http://vtun.sourceforge.net. The version of *vtun* that we will be using is part of the *BUILDROOT* distribution. The book "Building Linux Virtual Private Networks (VPNs)" by Oleg Kolesnikov and Brian Hatch has a complete chapter (8) on configuring and running vtun. Vtun was chosen because it has support for secure transfers using SSL, improved error handling (I/O timeout retries), and does not restrict the type of data and file systems that it serves. This allows us to use the same package for both the *isolinux* boot CD-rom images and root file systems for the routers. Unlike some other packages that were evaluated, *vtun* does not require patches to the Linux kernel.

- *kexec* is a set of kernel patches and user mode tools that allow you to load a new linux kernel from a currently executing one. It is being actively developed and maintained by Eric Biederman and the version that we are using can be found at: http://www.xmission.com/~ebiederm/files/kexec/2.6.8.1-kexec3/2.6.8.1-kexec3.gz and the user mode tools are at: http://www.xmission.com/~ebiederm/files/kexec/kexec-tools-1.8.tar.gz.

- *mkisofs* is part of a suite of utilities to create CD-rom images written by Joerg Schilling. It takes a directory tree located on disk a produces an ISO-9660 standard CD-rom image. The version of *mkisofs* that we will use is part of the Gentoo distribution (*emerge app-cdr/cdrtools)*.

- *OpenSSH* (home page: http://www.openssh.org) is the widely used open source implementation of a SSH server and client. The software will be downloaded and configured by the *BUILDROOT* script.

- *openSSL* (home page: http://www.openssl.org) is the secure session layer (SSL) library that is linked into the *vtun* and *OpenSSH* utilities to encrypt the network data streams. The version (0.9.7e) of openSSL that we will use is also managed by the

---

[10] Static linking is when all of the support (library) routines required by an executable program are included directly in the program's binary file instead of a "stub" routine that will load the routines dynamically at runtime. One mechanism that can be used to subvert security on UNIX-like systems is to replace system routines in dynamic load libraries.

BUILDROOT script.

In order to incorporate the remote boot service into the network infrastructure, resources must be made available to store the boot images. Ideally, the system that houses the images and *vtund* server daemons should be dedicated solely to this function and not directly accessible from the internet. This isolation is to help minimize the risk of the boot image server becoming compromised. It will also mitigate the effects if the server does become compromised.

If an isolated dedicated system is not available, then using some sort of Linux virtualization technology should be given serious consideration. The most widely known package of this technology is a commercial product called VMware (home page: http://www.vmware.com). VMware provides the capability to run unmodified guest operating systems on an Intel x86 Linux or Windows host. In addition to VMware, there are two open source solutions that also show some promise: *UML* (User-mode Linux, home page: http://user-mode-linux.sourceforge.net) and Cambridge University's *Xen* (home page: http://www.cl.cam.ac.uk/Research/SRG/netos/xen). The major drawback to these two packages is that they require modification to the guest operating system.

For this study we will use VMware as the development, image, logging, and test platforms. The base operating systems for both the development and server platform is the hardened version of Gentoo Linux described earlier. Any configuration changes to the base system will be described below.

The remote boot service plan is implemented in of three phases: 1) creating a secured development environment, 2) creation and testing of the boot image environments (both initial and production images), and 3) burning the initial boot image to read-only media and its distribution. The following assumes a working knowledge of Linux/Unix commands and procedures.

Creating a secured development environment

To create a secure development environment, we will use as a base the hardened Gentoo system described in the first section of this paper. We will load this version into a VMware virtual machine. The following is an annotated checklist of the development system's installation and configuration (user inputs are shown in *italics*):

Initial Installation of Base operating system

1) (___) As the root user, create a new VMware virtual machine called "develop" using the standard Gentoo base system:

```
# mkdir /vmware/develop; cd /vmware/develop
# tar xvpf /vmware/images/vmware-gentoo-2.6.x.tgz
```

The above will load the previously saved VMware virtual machine image into a new directory called “develop”. We next have to change the file names of the disk image and VMware configuration files to match the new virtual machine name.

```
# mv gentoo-s001.vmdk develop-s001.vmdk
# mv gentoo-s002.vmdk develop-s002.vmdk
# mv gentoo-s003.vmdk develop-s003.vmdk
# mv gentoo.vmdk develop.vmdk
# mv gentoo.vmx develop.vmx
```

We now need to edit the virtual machine’s configuration and disk descriptor file to refer to the new file names. Use the *ed* text editor to perform the following changes:

```
# ed develop.vmdk
g/gentoo-s00/s//develop-s00/g
w
q

# ed develop.vmx
g/gentoo.vmdk/s//develop.vmdk/g
w
q
```

2) (___) Once the changes to the configuration files have been made, start the virtual machine by issuing the following command and clicking on the “start this virtual machine” option:

```
# vmware develop.vmx
```

A pop-up window will appear saying that the location of this virtual machines configuration files have changed since it was last powered on. Make sure that the “Create a new identifier” option is selected and click on the “OK” button. The new virtual machine will boot.

3) (___) At the “gentoo login:” prompt: enter *root* and press return.
4) (___) At the “password:” prompt enter *default%pass*.
5) (___) At the “gentoo root#” prompt:

Enter the following command to change the root password:

```
gentoo root # passwd
New UNIX password: a new secure password
Retype new UNIX password: re-enter the new password
passwd: password updated successfully
```

The above will change the root password, please choose a password that is secure. A good method would be to choose a random string or at least 8 characters with the following characteristics: at least 2 upper case letters, at least 2 digits, at least 2 lower case letters, and at least 1 punctuation character. They should be in a random order and not used anywhere else. IT SHOULD NOT BE THE SAME AS THE ROOT PASSWORD FOR THE SYSTEM RUNNING VMWARE.

6) (___)Change the host name of the VMware virtual machine:

Edit the hostname found in the file: /etc/hostname:

```
develop root # ed /etc/hostname
s/gentoo/develop
w
q
```

7) (___)Make sure that the virtual machine image is up to date:

```
develop root # emerge sync
develop root # emerge –uaD world
```

The two *emerge* commands will update the gentoo image with GIAC University's local gentoo repository. They will download and install any packages that have changed.

8) (___)The initial base system install has been completed. At this point you can shut the virtual machine down by issuing a *shutdown –h now* command or you may continue with installing the development packages (step 10 in the next section).

Installation of Required Development Packages

In this section we will download, configure, and install the software packages that we will use to create our custom boot images. Parts of this section were developed using material from the "Gentoo Linux LiveCD for Dummies!" mini-how to document[11] and the buildroot, Busybox & uClibc documentation.[12] Note: The URLs used were correct as of March 10, 2005.

9) (___)If you are not logged into the root account of the development virtual

---

[11] This document is part of the Gentoo Documentation, Tips & Tricks forum and can be found at: http://forums.gentoo.org/viewtopic-t-244837.html.

[12] The buildroot documentation can be found at http://buildroot.uclibc.org/buildroot.html. The uClibc FAQ is at http://uclibc.org/FAQ.html and the BusyBox document is at http://busybox.net/downloads/BusyBox.html.

machine, do so now.

10) (___) Change to the source directory

```
develop root # cd /develop/src
```

11) (___) Download the necessary packages using the *wget* utility:

```
develop src # wget http://www.xmission.com/~ebiederm/files/kexec/2.6.8.1-kexec3/2.6.8.1-kexec3.gz
develop src # wget http://www.xmission.com/~ebiederm/kiles/kexec/kexec-tools-1.8.tar.gz
develop src # wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.8.1.tar.gz
```

12) (___) At this point, the /develop/src directory should like this:

```
develop src # ls –l /develop/src
total 48724
-rw-r--r--  1 root root       23484 Aug 19 2004  2.6.8.1-kexec3.gz
-rw-r--r--  1 root root       40978 Dec  1  2002  kexec-tools-1.8.tar.gz
-rw-r--r--  1 root root 44683888 Aug 14  2004 linux-2.6.8.1.tar.gz
```

13) (___) Merge the standard Gentoo packages that we also need to create ISO 9660 images:

```
develop src # emerge app-cdr/cdrtools
develop src # emerge sys-boot/syslinux
```

14) (___) Untar and patch version 2.6.8.1 of the Linux kernel:

```
develop src # cd /develop/src
develop src # tar xzpf /develop/src/linux-2.6.8.1.tar.gz
```

Apply the kexec patch to the linux kernel:

```
develop src # cd linux-2.6.8.1
develop src # gunzip –c < /develop/src/2.6.8.1-kexec3.gz | patch –p1
```

15) (___) Configure the Linux kernel (version 2.6.8.1 with our patches)

This step will customize the Linux kernel for our router. Our configuration changes remove most of the “normal” configuration options from our kernel. This limits the exposure to security vulnerabilities of our kernel (an attacker can’t use a buffer overflow exploit in the sound card driver, if the driver isn’t configured into the system).

To start the kernel configuration:

```
develop src # cd /develop/src/linux-2.6.8.1
develop linux-2.6.8.1 # make menuconfig
```

The system will display of tree based set of configuration menus. Make the following changes (DO NOT CONFIGURE THESE AS MODULES, BUILD THEM INTO THE KERNEL INSTEAD):

```
Loadable module support
        DISABLE Enable loadable modules support
        DISABLE         all other module options

Processor type and features
        ENABLE          Symmetric multi-processing support
        ENABLE          SMT (Hyperthreading) scheduler support
        ENABLE          kexec system call (EXPERIMENTAL)
```

The symmetric multi-processing and SMT support is enabled because the hardware platform that we are using (Dell PowerEdge 2650) has dual Hyperthreaded Pentium Xeon processors. Enabling the kexec system call will activate the kexec patch that we applied earlier. Kexec will be used to transfer control from the initial boot kernel to the production kernel loaded from the remote server.

```
Block devices
        ENABLE Loopback device support
```

Here we enable the loopback device support so we can mount the root file system contained on the initial boot medium. The enhanced network device is the kernel device driver used to communicate with our remote server to load the production system.

```
ENABLE Ramdisk support
(16384) Default RAM disk size (Kbytes)
ENABLE Initial Ramdisk (initrd) support
```

This turns on the kernel support that we need to load our initial boot system and production systems.

```
ATA/ATAPI/MFM/RLL Support
        DISABLE Include IDE/ATA-2 Disk support
        ENABLE  Include IDE/ATAPI CDROM support

SCSI device support
        DISABLE legacy /proc/scsi/ support
        DISABLE SCSI disk support
```

The above will disable kernel support for local disks. Since we will be running everything from the network block device, the only local device that we need is the CD-ROM drive (/dev/hdc).

Networking support
Networking options →
    Network packet filtering →
        IP Netfilter Configuration →
            DISABLE connection tracking
            ENABLE IP tables support

DISABLE Dummy net driver support

We configure the network and firewall support at this point. Depending on the specific router configuration additional settings on the netfilter configuration will be required. We will only concern ourselves with firewall settings that are specific to protect the router itself. General firewall configuration is beyond the scope of this document.

Character devices
    ENABLE /dev/nvram support
    ENABLE Enhanced Real Time Clock Support

We configure in the /dev/nvram so we can use its contents as part of the session keys for the network disk driver.

Sound
    DISABLE Sound Card Support
USB Support
    DISABLE Support for Host-side USB

We disable both the sound and usb support in our kernel since we don't need them. If we were going to boot from a USB thumb drive, we would need to enable the USB support.

File Systems
    ENABLE Second extended FS support
    ENABLE Ext3 journaling file system support
    ENABLE Ext3 extended attributes
    DISABLE kernel automounter version 4 support

We need ext2 for our initial boot image and ext3 for possible use in the production kernel.

CD-Rom/DVD File systems
    ENABLE ISO 9660 file system support

ISO 9660 support is needed to boot from CD-rom

Pseudo filesystems
    ENABLE /proc file system support
    ENABLE /dev file system support

This provides the linkage to the kernel drivers from user mode programs.

Network file systems
    DISABLE NFS file system support
    DISABLE NFS Server support

We do not need to access or provide any NFS file systems.

We have finished the configuration. Save the configuration file and then build the kernel by entering:

```
develop linux-2.6.8.1 # make
```

The compiled kernel is now located in /develop/src/linux-2.6.8.1/arch/i386/boot/bzImage.

16) (___) Build the uClibc/Busybox (BUILDROOT) development environment

Issue the following commands to build the development environment. Note: this step requires the use of the subversion (svn) utility to download the buildroot distribution from the project's website. If svn is not installed, you must issue the following command: *emerge subversion.*

```
develop linux-2.6.8.1 # cd /develop/src
develop src # svn co svn://uclibc.org/trunk/buildroot
develop src # cd /develop/src/buildroot
develop src # make menuconfig
```

During the configuration phase, choose the default values with the following exceptions:

- Target Architecture →
  - i386
- Toolchain Options →
  - Kernel Headers →
  - select Linux 2.6.8 headers (with mips fixes)
  - Enable large file (size > 2GB) support
- Package Selection for the Target
  - select openssh
  - select openssl
  - select tcpdump
  - select vtun
  - select zlib
- Target Options →
  - select ext2 root filesystem for the target device
  - enter (20480) size in blocks
  - enter (2048) inodes.

Save the configuration file, and then issue a make command:

```
develop buildroot # make
```

This will build the development environment and a template root file system (located in the file root_fs_i386.ext2).

17) (___) Activate the new development toolchain

```
develop buildroot # cd /develop
develop develop # export \
        PATH=$PATH:/develop/src/buildroot/build_i386/staging_dir/bin
```

18) (___) Build the kexec tools using the new development chain:

```
develop develop # cd /develop/src/kexec-tools-1.8
```

Before making the tools, you need to change the name of the gcc compiler inside of the Makefile, change it to i386-linnux-gcc on both of the CC and CPP lines. The modified Makefile should look like:

```
OBJDIR:= ./objdir

CC:=i386-linux-gcc
LD:=ld
AR:=ar
AS:=as
CPP:= i386-linux-gcc –x assembler-with-cpp –E

include Makefile.main
```

Now, build the tools with:

```
develop kexec-tools-1.8 # make
```

19) (___) Create a custom version of BusyBox. Make sure that the following options are selected during the menuconfig:

```
Build Options →
        ENABLE Build BusyBox as a static binary (no shared libs)
Init Options →
        DISABLE Support running init from within an Initrd
```

Note: we will still be running init from within an initrd. If we do not disable this option, BusyBox will create a link to linuxrc instead of letting us create our own in our make_initial.sh and make_prod.sh scripts.

```
develop # cd /develop/src/buildroot/build_i386/busybox
develop busybox # cp .config busybox.config-develop
develop busybox # make menuconfig
```

The same *BUILDROOT* tree (with different configurations) is used by both the initial boot and production image creation process. An easy way to manage the different configurations for the initial and production phases is to save the files */develop/src/buildroot/.config* and */develop/src/buildroot/build_i386/busybox/.config* for both. Assuming that the files have been saved in /develop, to build the initial boot image configuration issue the following:

```
# cd /develop/src/buildroot
# cp /develop/initial.buildroot.config .config
# cp /develop/initial.busybox.config ./build_i386/busybox/.config
# cd ./build_i386/busybox/
# make oldconfig
# make clean
# cd ../..
# make oldconfig
# make
```

The production configuration can similarly build by using the same commands and substituting *prod.buildroot.config* and *prod.busybox.config* where appropriate.

Creating the Production Boot Image

All of the steps necessary to create the production boot image are contained in the *make_prod.sh* shell script. This script requires that the development environment be previously installed. There are two files produced by this script: a gzipped root file system containing the configured production system and an ISO CD-Rom image that can be used for testing.

```
1) #! /bin/sh
2) ISOLINUX=/usr/lib/syslinux/isolinux-debug.bin
3) INIT=/linuxrc     # use /linuxrc for live images, /bin/sh for
                             testing
```

Lines 2 and 3 specify that we want to use the debugging version of the isolinux boot loader and that the first program that will be executed by the initial boot kernel will be /linuxrc.

```
4) if [ -z "$1" ]
5) then
6) echo "Usage: $0 <router-IP>"
7) exit 1
8) fi
```

Lines 4 though 8 checks to make sure that the IP number of the router we are generating the system for is specified on the script command line.

```
9) cd /develop/images
10)      rm -rf $1-prod
11)      mkdir -p $1-prod/boot/isolinux
```

Now we create an empty directory tree to hold the production system.

```
12)      cp /develop/src/linux-2.6.8.1/arch/i386/boot/bzImage \
         $1-prod/boot/isolinux/vmlinuz
13)      cp $ISOLINUX $1-prod/boot/isolinux/isolinux.bin
```

Copy the kernel and boot loader that we want the production system to run.

```
14)      cat > $1-prod/boot/isolinux/isolinux.cfg <<END
15)      default vmlinuz
16)      append init=$INIT initrd=rootfs.gz root=/dev/ram0 rw debug
   BOOT_IMAGE=initial
17)      prompt 1
18)      timeout 300

19)      label vmlinuz
20)      kernel vmlinuz
21)      append init=$INIT initrd=rootfs.gz root=/dev/ram0 rw debug
   BOOT_IMAGE=initial
22)      END
```

Lines 14 through 22 will create the isolinux boot loader configuration file that will be written to the initial boot CD-Rom. This configuration will boot using the initird from the CD-Rom loaded into a ram disk.

```
23)      mkdir -p /tmp/$1-rootfs
24)      mount -t ext2 -o loop /develop/src/buildroot/prod.ext2 \
25)      /tmp/$1-rootfs
```

Mount the production root file system so that we can customize it for our specific router.

```
26)      cp $1-prod/boot/isolinux/vmlinuz /tmp/$1-rootfs/vmlinuz
27)      cat > /tmp/$1-rootfs/etc/inittab <<END_INITTAB
28)      ::sysinit:/bin/mount -t proc none /proc
29)      ::sysinit:/bin/mount -t tmpfs none /tmp
30)      ::sysinit:/bin/mount / -n -o remount,rw
31)      ::sysinit:/sbin/klogd
32)      ::sysinit:/sbin/syslogd -C
33)      ::sysinit:/bin/hostname $1
34)      ::sysinit:/sbin/ifconfig eth0 $1
35)      tty1::respawn:/sbin/getty 38400 tty1 linux
36)      END_INITTAB
```

Line 26 copies the kernel onto the root file system for the initial boot media. We also will create the inittab file that when /sbin/init executes will mount /proc, /tmp, and the root file systems in read/write mode. The system logging daemons are started. The network management interface and console terminal is also started. Since this is an example script, the additional network interfaces of the router are not shown, but are also started at this point. NOTE: To start up the OpenSSH daemon so we can access the router from the network management interface, a line running the SSH init script would be added after line 34 above (since the configuration of OpenSSH daemon is a common task, it is not included here).

```
cat > /tmp/$1-rootfs/etc/passwd <<END_PASSWD
root:x:0:0:root:/root:/bin/sh
sshd:x:22:22:sshd:/dev/null:/bin/false
netops:x:1000:100::/home/netops:/bin/sh
END_PASSWD
cat > /tmp/$1-rootfs/etc/shadow <<END_SHADOW
root:\$1\$6gCBOeed\$nfjrUg1d8Sg2wC3N8QOej/:12838:0:::::
sshd:*:9797:0:::::
netops:\$1\$t5IzqQmV\$nV1gdOJHOujf7WADpJ2D1/:12771:0:99999:7:::
END_SHADOW
```

Now create a minimal password file for the production system (these entries should be unique for each router and could be generated by another script or program). NOTE: It is important to escape any shell metacharacters like the dollar signs in the passwords or you will not be able to login to the system when it boots.

```
mkdir -p -m 711 /home/netops
chown 1000:100 /home/netops
chmod 444 /tmp/$1-rootfs/etc/passwd
chmod 400 /tmp/$1-rootfs/etc/shadow
```

Create an empty home directory for the network operations account and make sure that the permissions of the directory and the password files are correct.

```
cat > /tmp/$1-rootfs/etc/fstab <<END_FSTAB
/dev/ram0   /     ext2  defaults    0 0
END_FSTAB
chmod 644 /tmp/$1-rootfs/etc/fstab
```

Create a template /etc/fstab file. This is necessary so we can remount the root file system in read/write mode when we boot.

```
rm -f /tmp/$1-rootfs/linuxrc
ln /tmp/$1-rootfs/bin/sh /tmp/$1-rootfs/linuxrc
```

Make sure that /linuxrc utility is correctly linked to the BusyBox binary. This will cause the BusyBox program to act as the init process when the system boots.

```
umount /tmp/$1-rootfs
rmdir /tmp/$1-rootfs
```

Unmount the production root file system and clean up the temporary mount point.

```
gzip -c -9 < /develop/src/buildroot/prod.ext2 ./$1-prod.fs
cp ./$1-prod.fs ./$1-prod/boot/isolinux/rootfs.gz
```

Compress the root file system with gzip and copy it where the isolinux configuration file expects to find it during boot.

```
mkisofs -o $1-prod.iso -b boot/isolinux/isolinux.bin \
-c boot/isolinux/boot.cat -no-emul-boot -boot-load-size 4 \
-boot-info-table ./$1-prod
```

```
64)      exit 0
```

Finally, we make an ISO-9660 CD-Rom image that can be used for testing. The CD-Rom image is not used other than testing. The compressed root file system is downloaded by the initial boot image and booted instead. See the Initial Boot Media section for more information.

The process to build the production image for the 192.168.139.64 router would look like this:

```
develop root # cd /develop
develop develop # ./make_prod.sh 192.168.139.64
Size of boot image is 4 sectors -> No emulation
Total translation table size: 2048
Total rockridge attributes bytes: 0
Total directory bytes: 4096
Path table size(bytes): 38
Max brk space used 0
4660 extents written (9 MB)


develop develop # ls /develop/images
192.168.139.64-prod     192.168.139.64-prod.iso        192.168.139.64-
prod.fs
develop develop #
```

The results of the *./make_prod.sh* script are the directory 192.168.139.64-prod, the ISO-9660 CD-Rom image 192.168.139.64-prod.iso, and the compressed root file system 192.168.139.64-prod.fs. Only the 192.168.139.64-prod.fs is required for the proper functioning of the remote boot process.

Configuring the Image Server

The following steps describe the procedure used to configure the server(s) that our routers will contact to download their production system images. We will use the *vtun* virtual tunneling package to securely transfer information between the server and routers. Multiple *vtund* daemons can be installed on a single system. However, they each should use a different port and storage directory for each daemon.

25) (___) Install the base operating system on the system that will be acting as the image server. This could be the same procedure as described in steps 1 through 8 above (with appropriate changes for host name and IP numbers). Since *vtund* daemons run in user-mode with non-privileged ports, no special kernel patches or configuration are needed.

26) (___) Copy the *vtund* binary from the development machine over to the image server. The *vtund* daemon was built as part of the *BUILDROOT* process described earlier.

27) (___) For each of the client systems (routers) run the *make_prod.sh* and *make_initial.sh* scripts (unless you have already done so).

28) (___) On the logging server (in this study, it is the same machine as the image server, but it could be another system). Make sure the syslogd daemon is active.

29) (___)For each client system create the *vtun* configuration files. For each client, two separate configuration files are needed, one for the operating system image and the other for the system logging messages. Below is an example *vtun* configuration file to serve the OS image for the 192.168.139.64 router. The configuration file to handle the system logging is very similar and is not included here (for space reasons).

```
options {
  port 5000;
  timeout 60;
}

Download-image {
  type pipe;
  pass 45566cd2864efa36ad03eab72fbe5808;
  encrypt yes;
  up {
    prog /bin/bash "-c '/bin/dd
      if=/develop/images/192.168.139.64-prod.fs'";
  };
}
```

NOTE: The “prog” line and the one following it are displayed as two lines but is a single line in the actual file. The password is uniquely generated for each client system and is an MD5 checksum of system specific information. See the section “Generating Passwords” below for more information on how this value is computed.

30) (___) On both the image and logging servers, configure the system to automatically start the *vtund* server daemons for each client system. An easy way to do this is to create a shell script called */etc/init.d/vtund* that correctly responds to start, stop, and restart arguments. On a Gentoo system, you can use the “*rc-update add vtund default*” command to do this. See the *vtun* and Gentoo documentation for information.

31) (___) Verify that all of the *vtund* server daemons start up correctly by issuing a “*/etc/init.d/vtund start*” command. If you issue a “*ps agx*” command, all of the *vtund servers* should appear. Also, a “*netstat –an*” command should show all of the ports used by the daemons.

32) (___) Any host or network based firewall configuration changes that are necessary to allow access to the image and logging servers should be done

at this time. The actual steps necessary to accomplish this are beyond the scope of this document.

Creating the Initial Boot Image

As with the *./make_prod.sh* script, the initial boot image is produced by executing the *./make_initial.sh* script. This script also requires that the development environment be previously installed. The result of the *.make_initial.sh* script is an ISO-9660 CD-Rom image that when booted, will connect to the image server, download the production OS image and start it running.

```
1) #! /bin/sh
2) ISOLINUX=/usr/lib/syslinux/isolinux-debug.bin
3) INIT=/linuxrc     # use /linuxrc for live images, /bin/sh for
   testing
4) SERVER_IP="192.168.139.11"
5) SERVER_PORT=5000
```

In lines 1 though 5, we setup several environment variables that will be used throughout the rest of the script. These specify the version of the *isolinux* boot loader, system init program, and the IP number & port of the image server.

```
6) if [ -z "$1" ]
7) then
8)   echo "Usage: $0 <router-IP>"
9)   exit 1
10)     fi
```

Make sure that we have the client system's IP number as an argument on our command line.

```
11)     if [ ! -f /develop/src/kexec-tools-1.8/objdir/build/sbin/kexec
   ]
12)     then
13)       echo " *** kexec utility not found ***"
14)       exit 1
15)     fi
```

Make sure that the *kexec* utility that we will need to switch to the production kernel has been built.

```
16)     cd /develop/images
17)     rm -rf $1-initial
18)     mkdir -p $1-initial/boot/isolinux
```

Create an empty directory tree to hold the router's *isolinux* boot loader configuration files.

```
19)     cp /develop/src/buildroot/initial.ext2 $1-
   initial/boot/isolinux/rootfs
```

```
20)     cp /develop/src/linux-2.6.8.1/arch/i386/boot/bzImage \
                $1-initial/boot/isolinux/vmlinuz
21)     cp $ISOLINUX $1-initial/boot/isolinux/isolinux.bin
```

Copy the root file system, Linux kernel, and the CD-Rom boot loader that will be used by the initial boot image.

```
22)     cat > $1-initial/boot/isolinux/isolinux.cfg <<END
23)     default vmlinuz
24)     append init=$INIT initrd=rootfs.gz root=/dev/ram0 rw debug
   BOOT_IMAGE=initial
25)     prompt 1
26)     timeout 100

27)     label vmlinuz
28)     kernel vmlinuz
29)     append init=$INIT initrd=rootfs.gz root=/dev/ram0 rw debug
   BOOT_IMAGE=initial
30)     END
```

Copy the *isolinux* configuration file for the CD-Rom.

```
31)     mkdir -p /tmp/$1-rootfs
32)     mount -t ext2 -o loop $1-initial/boot/isolinux/rootfs /tmp/$1-
   rootfs
```

Create a temporary mount point and mount the initial boot root file system on it.

```
33)     cp /develop/src/kexec-tools-1.8/objdir/build/sbin/kexec \
              /tmp/$1-rootfs/sbin/kexec
```

Copy the *kexec* utility program over to the root file system. Now we create the startup script that will be the first thing that is executed when the initial boot media is loaded. Note: if this file is replaced by a link to */bin/sh* (see line 3 above) the ISO image produced can be used for testing and password generation purposes by starting a shell session after booting.

```
34)     cat > /tmp/$1-rootfs/linuxrc <<ENDLINUXRC
35)     #! /bin/sh
36)     # automatically configure the GIAC University router $1
37)     # eth0 by convention is always the network management interface

38)     mount -t proc none /proc
39)     mount -t tmpfs none /tmp    # we need to create /tmp files

40)     /sbin/klogd
41)     /sbin/syslogd -C
42)     /sbin/ifconfig eth0 $1            # bring up our netmgt
   interface

43)     cat >/tmp/download <<END
44)     #! /bin/sh
45)     /usr/bin/tee /tmp/remote.fs | /bin/gunzip -c -f | /bin/dd
   of=/dev/ram1
46)     /bin/mount -t ext2 /dev/ram1 /mnt
```

```
47)     /sbin/kexec -f /mnt/vmlinuz --init=/linuxrc --
   initrd=/tmp/remote.fs \\
              --append="root=/dev/ram0 init=/linuxrc"
48)     exit 0
49)     END
```

Lines 38 and 39 will cause the /proc and /tmp file systems to be mounted during boot. Lines 40 through 42 will start the kernel & system logging daemons and configure the network management interface. Lines 43 through 49 will create a file called /tmp/download that will be called when the production image is downloaded to a Ram disk on the router. This is done by the /linuxrc script when the initial boot media is booted on the router. After the production root file system is loaded to /dev/ram1, the *kexec* utility is executed to reboot the system into the new (production) kernel. Since we do not have any local file systems, we can do a “forced” shutdown (-f option on the *kexec* command). If this option was omitted, *kexec* would try and execute the shutdown command before rebooting. If the *kexec* fails, line 48 will cause a kernel panic and cause it to hang (interrupting normal network service).

```
50)     /bin/chmod 500 /tmp/download
51)     cat >/tmp/vtund-initial.cnf <<END_Download
52)     options {
              port $SERVER_PORT;
              timeout 30;
53)     }

54)     Download-image {
55)      type pipe;
56)      pass \`/sbin/ip addr | /bin/cat - /dev/nvram | /usr/bin/md5sum
   |               /usr/bin/cut -f 1 -d ' '\`;
57)      encrypt yes;
58)      up {
59)        prog /tmp/download;
60)      };
61)     }
62)     END_Download
```

Create the client side *vtun* configuration file that will try and download the production OS image. The shell pipeline found in line 56 will calculate the password to be used by *vtun* to download the image.

```
63)     /bin/chmod 400 /tmp/vtund-initial.cnf
64)     /usr/sbin/vtund -f /tmp/vtund-initial.cnf Download-image
   $SERVER_IP
```

Start up the client side virtual tunnel to download the production image after making sure that the permissions are set securely on its configuration file.

```
65)     /bin/sleep 40

66)     echo -e "\n\n\t*** Production image download failed ***\n\n"
67)     exit 0
68)     ENDLINUXRC
69)     chmod 500 /tmp/$1-rootfs/linuxrc
```

The *sleep* command found on line 65 causes the system to wait for the production image download to complete and begin execution. If the transfer (or *kexec* call) fails, the sleep timer will expire and the echo and exit commands will be executed, causing a kernel panic.

```
70)     umount /tmp/$1-rootfs
71)     rm -rf /tmp/$1-rootfs
```

Unmount the initial boot media file system and clean up the temporary mount point.

```
72)     gzip -9 ./$1-initial/boot/isolinux/rootfs
73)     mkisofs -o $1-initial.iso -b boot/isolinux/isolinux.bin \
              -c boot/isolinux/boot.cat -no-emul-boot -boot-load-size 4
              \
              -boot-info-table ./$1-initial
74)     exit 0
```

Compress the initial boot media root file system, copy it over to the *isolinux* configuration directory tree and generate the ISO-9660 CD-Rom image file.

33) (___) Securely distribute the CD-Rom and load it into the remote router's CD-Rom drive. Once the CD-Rom has arrived at the remote site, log onto the router and umount the old CD-Rom and replace it with the new version. Then "relock" the drive by issuing a mount command similar to "*mount –t iso9660 –r /dev/cdrom /mnt/cdrom".* The new version of the boot media will not become active until the router is rebooted or other specific action is taken.

Generating Passwords

One of the key questions for this project is: how should the passwords used to authenticate data transfers to the router be generated?

Placing a plain-text pass phrase on the initial boot media is insecure; an attacker can simply look at the CD-Rom contents and find out what it is. Another possibility is to require someone to manually enter the pass phrase on the keyboard. But this conflicts with the project goal of not requiring staff at the remote site on an ongoing basis.

A simple solution is to generate a password based on hardware specific information. For the initial boot media, we can generate a MD5 checksum based on the network management MAC address and the /dev/nvram device[13]. For the production boot image, we will use the MD5 checksum of the MAC address and the initial boot CD-Rom. While the password may still be determined if an attacker knows this fact and has access to the hardware, it is a reasonable compromise. This pass phrase will

---

[13] The nvram device allows programs to access the contents of the system's BIOS. Also, since we are not using any local disk drives, entering custom BIOS disk drive settings can introduce some variation.

`only be used` during the download phase. If this password needs to be used for anything other than an occasional boot, a better method of securing it would be needed.

Before a router is deployed in the field, an IP number for its network management interface must be assigned and its MD5 checksums performed. These values are then entered as the “pass” values of the corresponding image and logging *vtun* configuration files.

34. (___) Compute the router’s MD5 checksum, using the following command:

```
/# /sbin/ip addr | /bin/cat - /dev/nvram | /usr/bin/md5sum | \
              /usr/bin/cut -f 1 -d ' '
853d553f9554442ba635766ad69d757
```

This command needs to be executed directly on the router hardware in order to obtain a system “fingerprint”. One method that can used to do this is to generate an initial boot media CD-Rom using */bin/sh* instead of */linuxrc* as the init program.

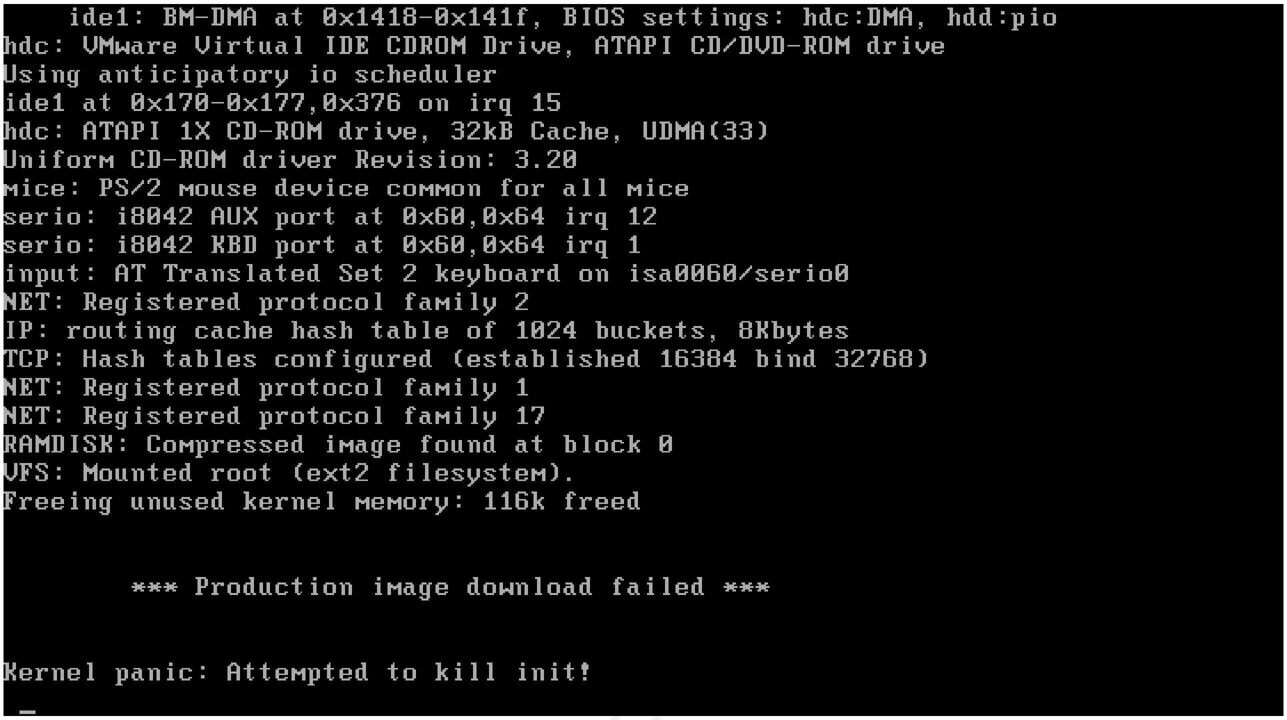
Verifying Correct Installation

Because of the way the *vtund* and *kexec* services are used by the initial boot process, verification that it has been correctly installed is simple: it either works or it doesn’t. If everything is installed correctly, the initial boot image will load the system and request the production image from the remote server. Since it is not a good idea to take the network down to test the initial boot image, using an isolated system or Vmware session is a good idea.

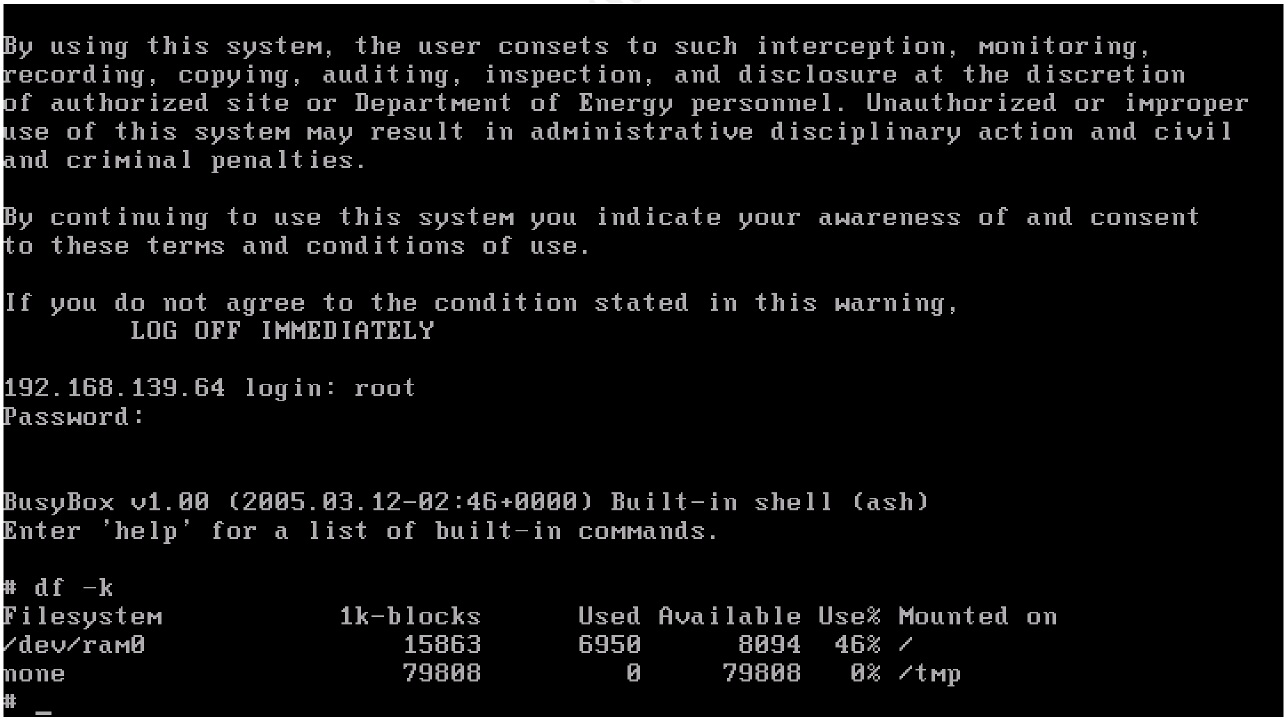
Testing the production image for correct installation of the logging server is equally simple. After the router has booted the production image, simply log onto it and log some messages to the syslogd. If they show up on the logging server, it is installed correctly. Of course, this only verifies the function of the network boot service, additional software on the production image should have similar verification procedures.

The following has been extracted from the image server system log showing both a successful and unsuccessful router boot. Three screenshots showing both the failure to load the production image and the output of the ps and df commands of the production image actually running on the router are also included.

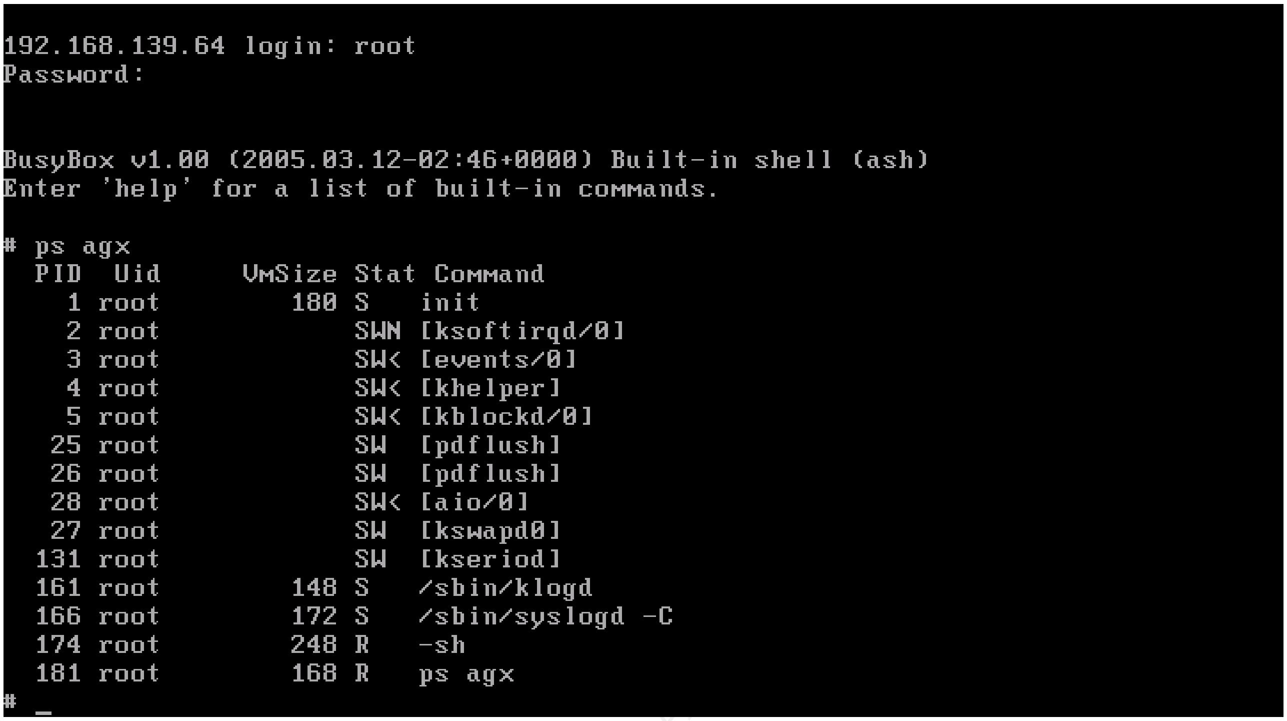
```
Mar 12 21:51:23 develop vtund[5932]: VTUN server ver 2.6 03/11/2005 (stand)
Mar 12 21:57:08 develop vtund[5938]: Session Download-image[192.168.139.64:32768] opened
Mar 12 21:57:08 develop vtund[5938]: BlowFish encryption initialized
Mar 12 21:57:15 develop vtund[5938]: Session Download-image closed

Mar 12 22:21:52 develop vtund[6061]: VTUN server ver 2.6 03/11/2005 (stand)
Mar 12 22:22:35 develop vtund[6063]: Session Download-image[192.168.139.64:32768] opened
Mar 12 22:22:35 develop vtund[6063]: BlowFish encryption initialized
Mar 12 22:22:35 develop vtund[6066]: Denied connection from 192.168.139.64:32769
```



```
    ide1: BM-DMA at 0x1418-0x141f, BIOS settings: hdc:DMA, hdd:pio
hdc: VMware Virtual IDE CDROM Drive, ATAPI CD/DVD-ROM drive
Using anticipatory io scheduler
ide1 at 0x170-0x177,0x376 on irq 15
hdc: ATAPI 1X CD-ROM drive, 32kB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.20
mice: PS/2 mouse device common for all mice
serio: i8042 AUX port at 0x60,0x64 irq 12
serio: i8042 KBD port at 0x60,0x64 irq 1
input: AT Translated Set 2 keyboard on isa0060/serio0
NET: Registered protocol family 2
IP: routing cache hash table of 1024 buckets, 8Kbytes
TCP: Hash tables configured (established 16384 bind 32768)
NET: Registered protocol family 1
NET: Registered protocol family 17
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 116k freed


        *** Production image download failed ***


Kernel panic: Attempted to kill init!
 _
```



```
By using this system, the user consets to such interception, monitoring,
recording, copying, auditing, inspection, and disclosure at the discretion
of authorized site or Department of Energy personnel. Unauthorized or improper
use of this system may result in administrative disciplinary action and civil
and criminal penalties.

By continuing to use this system you indicate your awareness of and consent
to these terms and conditions of use.

If you do not agree to the condition stated in this warning,
        LOG OFF IMMEDIATELY

192.168.139.64 login: root
Password:


BusyBox v1.00 (2005.03.12-02:46+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                15863      6950      8094  46% /
none                     79808         0     79808   0% /tmp
# _
```



```
192.168.139.64 login: root
Password:


BusyBox v1.00 (2005.03.12-02:46+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# ps agx
  PID  Uid     VmSize Stat Command
    1 root        180 S    init
    2 root            SWN  [ksoftirqd/0]
    3 root            SW<  [events/0]
    4 root            SW<  [khelper]
    5 root            SW<  [kblockd/0]
   25 root            SW   [pdflush]
   26 root            SW   [pdflush]
   28 root            SW<  [aio/0]
   27 root            SW   [kswapd0]
  131 root            SW   [kseriod]
  161 root        148 S    /sbin/klogd
  166 root        172 S    /sbin/syslogd -C
  174 root        248 R    -sh
  181 root        168 R    ps agx
# _
```

Ongoing Maintenance and Auditing plan

Having a well defined procedure for managing the software configuration and generation of the router boot images is an important part of any ongoing maintenance and auditing plan. Having detailed creation, installation, and testing procedures becomes part of the audit trail documenting the configuration to the router and image server.

The procedure for ongoing maintenance of the remote image service is very simple. Because of its implementation, many of the typical issues associated with maintenance simply go away. Since the initial boot media is read-only, we do not need to worry about file contents and permissions being altered in an insecure way. Verifying the MD5 checksum of the entire media is enough to check this (and this is an integral part of each boot of the router hardware). By design the network will “fail closed” in the event of a system reboot or other event that impacts the router’s network connectivity. This minimizes the risk of service failure going undetected.

Insuring the security of the systems that serves the production software images and logging requires little more effort than those needed to check its base operating system. Typical steps used to ensure that the base OS is secure, like running nmap scans to check for open network vulnerabilities and using tripwire for host file system changes are completely appropriate for the file server. The only thing that needs to be done is to include the *vtund* server daemons, configuration files, and image files in the nmap and tripwire evaluations.

You can make sure that the remote image server is functioning properly by reviewing the system logs and output from the *ps agx* command. In general, the recovery procedure will be to restart the server daemon for the effected image file. Of course, if the outage was caused by a successful attack, before restarting the daemon, the server must be restored to a known secured state. Normally, this is done by reinstallation of the base operating system, the *vtund* servers & configuration files, and the image files from known good backups and then applying any corrective measures. If the systems are configured to disable network access based on the image server failing, the steps necessary to re-enable network access should be done here.

Summary

This study describes a method to securely manage the network configuration of Linux based routers. It is intended for use in a semi-open office environment. Overall, the approach described in this study is feasible and has been successfully implemented in practice. The process is generic enough that it can also be used in most linux "appliance" uses such as thin clients, kiosks, file server, etc.

One additional area that needs further research is how the password for each router is managed. The current method should be considered insecure. It should be thought of only as a mechanism to help prevent the casual disclosure of the password by someone not actively attacking the system. An ideal solution to this issue would be to use some sort of hardware device that has anti-tampering features incorporated into it. If this device could provide two-factor or one-time pad authentication functionality similar to SecurID or s/key[14] without manual intervention, security could be dramatically improved.

References

In addition to the course materials, the following reference sources were used in the preparation of this paper:

1) *isolinux & syslinux,* SYSLINUX – The easy to use Linux Bootloader, H. Peter Anvin. Home page: http://syslinux.zytor.com. The version of isolinux that we will use is part of the Gentoo distribution (*emerge sys-boot/syslinux).*
2) *BUILDROOT,* Erik Andersen, Home page: http://buildroot.uclibc.org.
3) *uClibc*, Erik Andersen. Home page: http://www.uclibc.org.

---

[14] S/key was developed and AT&T Bell Labs (Bellcore). SecurID is a commercial product by RSA Security Inc. See http://www.csua.berkeley.edu/skey-howto.html and http://www.rsasecurity.com/node.asp?id=1156 for more information about s/key and SecurID.

4) *BusyBox,* Erik Andersen. Home page: http://busybox.net.
5) *Linux Kernel 2.6.8.1*, Linus Torvalds, Home page: http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.8.1.tar.gz.
6) *vtun*, "Virtual Tunnel", Maxim Krasnyansky and the Vtun team. Home page: http://vtun.sourceforge.net. The version of *vtun* that we will be using is part of the *BUILDROOT* package.
7) *Vtun,* "Building Linux Virtual Private Networks (VPNs)", Oleg Kolesnikov and Brian Hatch, New Riders Publishing, 2002, 231-263.
8) *kexec*, Eric Biederman. Download page: http://www.xmission.com/~ebiederm/files/kexec/2.6.8.1-kexec3/2.6.8.1-kexec3.gz
9) *kexec* user mode tool, Eric Biederman. Download page: http://www.xmission.com/~ebiederm/files/kexec/kexec-tools-1.8.tar.gz.
10) *mkisofs*, Joerg Schilling. The version of *mkisofs* that we will use is part of the Gentoo distribution (*emerge app-cdr/cdrtools)*.
11) *OpenSSH,* The OpenSSH project. Home page: http://www.openssh.org. The version of OpenSSH that we will be using is downloaded and configured by the BUILDROOT script.
12) *openSSL*, The OpenSSL project. Home page: http://www.openssl.org. The version (0.9.7e) of openSSL that we will use is downloaded and configured by the BUILDROOT script.
13) LinuxBIOS, The LinuxBIOS project. Home page: http://www.linuxbios.org.
14) Etherboot, The Etherboot Project. Home page: http://etherboot.sourceforge.net.
15) Gentoo Linux. The Gentoo Foundation. Project Home Page: http://gentoo.org.
16) Gentoo Linux/x86 Handbook. The Gentoo Foundation. Download page: http://www.gentoo.org/doc/en/handbook/index.xml.
17) Using the initial RAM disk (initrd), Werner Almesberger and Hans Lermen, This document is part of the standard documentation for the Linux kernel and can be found in the kernel source directory tree at Documentation/initrd.txt.
18) RSA SecurID Authentication, RSA Security, Inc. http://www.rsasecurity.com/node.asp?id=1156.
19) S/Key Howto. The FreeBSD Handbook, Garrett Wollman. http://www.csua.berkeley.edu/skey-howto.html.