Automation of Secure Debian/GNU Linux Installations with Fully Automatic Installation

By
Mathew A. Chrystal

June 18, 2004
GSEC Practical (v.1.4b)
Option 2

ABSTRACT

A Linux implementation project was undertaken at an unnamed university college to determine its feasibility for research computing. The project mandate was to explore a Linux deployment solution from initial installation and configuration to ongoing maintenance and security upgrades. Since university environments tend to be somewhat less secure than non-academic concerns, security would necessarily be an important component of the planned deployment. Because the computer network is controlled at the university level and is inaccessible to college personnel, security measures would need to be implemented at the host level. Due to the large number of computers involved, an automated installation was developed to standardize configurations for all installation clients. The results of the deployment seem to meet the computing needs of the college research community with minimal installation and maintenance overhead along with vastly improved security.

MOTIVATION

Shrinking operating budgets necessitated the exploration of Linux as a replacement for traditionally expensive proprietary UNIX computers. A general feeling also existed that advances to the UNIX platform were not keeping pace with rapidly improving commodity computers. The growing number of scientific software applications becoming available for Linux as well as it's free and open licensing standards also impacted the decision. Additionally the vast majority of Beowulf type compute clusters, which allow large and computationally complex problems to be solved in a fraction of the time that would be required by individual computers, are based on Linux.

Previous attempts to deploy Linux by the college were unsuccessful. Some of the contributing factors to the unsuccessful implementation were proper Linux system administration skills and the actual Linux distribution itself. Previous deployment was attempted at a time when Linux had many dangerous services enabled by default. Lacking rudimentary system administration skills these services were not secured properly which resulted in numerous hacked computers. These computers were being used to serve illegal software and for launching attacks on other computers and were disconnected by the university. Due to these reasons, a properly planned deployment was essential because these machines would be scrutinized very closely. During the initial planning stages, Redhat Linux was the chosen distribution. The main reasons for choosing Redhat were that it was the most common Linux distribution, many proprietary software programs were certified for Redhat, and its ability to perform automated installs using anaconda. In the interim between planning and deployment Redhat initiated short end of life cycles with their ultimate purpose being to charge for future releases of free software. Since cost was one of the main factors in the decision to try Linux, Redhat was precluded from future consideration. Due to this immediate and drastic reversal of policy a search was started to find another distribution for use in the deployment. This change in policy by Redhat caused semi-commercial distributions to be dropped from consideration. Finally debian was chosen as the distribution for deployment due to its social contract stating that debian will remain 100% free software.

PREPARATION

Hardware requirements for the computers were determined and a single machine was purchased for testing purposes. Since the actual deployment would potentially include a large number of publicly accessible computers, hardware and software security concerns needed to be addressed. The large number of installations required necessitated the development of an automated installation system to distribute and configure the operating system and related software. Debian includes a package called fai (fully automatic installation) which could perform the required automatic installations.

AUTOMATED INSTALLATIONS

Fai (fully automatic installation) is a collection of shell, perl, and cfengine scripts that perform automated installation and configuration of debian Linux on client computers (Lange, 2004). It is especially useful for environments which require large numbers of identically or similarly configured computers. By using classes, which are basically text configuration files defined in the installation process, fai maintains a high degree of flexibility and is almost infinitely configurable.

An installation server is required for fai to perform automated installations. The server must have debian as the operating system and packages fai and fai-kernels installed to provide the needed components of fai. Additionally, the server must contain a mirror of the debian distribution that is to be installed on the client computers. Two scripts mkdebmirror and debmirror are needed to create the local debian mirror. While debmirror is included in the debian distribution, the script mkdebmirror seems to have problems interacting with this particular version of debmirror. To ensure properly functioning scripts, both scripts should be downloaded from the fai website http://www.informatik.uni-koeln.de/fai. After configuring mkdebmirror with the proper parameters such as distribution, location of the debmirror script and preferred location of the local mirror, simply executing ‘mkdebmirror –progress’ will make the local mirror and print the progress to standard output. The next step is to configure the fai program.

The fai configuration file is located in /etc/fai/fai.conf and it must be edited to reflect your particular situation. Parameters such as installserver, boot kernel-image, and distribution can be defined in this file. The variable KERNELPACKAGE defines the kernel that will be booted on the initial client installation. The default kernel in the fai-kernels package as of this writing is kernel-image-2.4.24-fai_1_i386.deb which should be satisfactory for most applications. If another kernel has to be compiled due to missing drivers etc., it is important to note that CONFIG_NFS_ROOT option needs to be compiled into the kernel. The required ethernet drivers also must be compiled into the kernel and not compiled as kernel modules. Essentially everything needed for your initial install has to be compiled into the kernel. Kernel options compiled as modules generally will not work for the initial installation. It should be remembered that this kernel is only used to boot the install client; a different kernel can be added during the install process. It is also worth noting here that the /etc/fai/sources.list is the sources.list file that will be copied to the install hosts.

The installation server must also act as an nfs server so the install clients have access to the local debian mirror, nfsroot directory, and fai configuration files. Directories containing these files have to be exported to enable access by the clients during installation. Reasons for needing access to the installation distribution and fai configuration files are quite obvious but some explanation might be required for the nfsroot directory. Previously it was mentioned that one of the requirements of the installation kernel was that it had to have support for nfs mounting of a root partition (CONFIG_NFS_ROOT=y). This is required because the installation clients need to mount this directory as their root directory for bootstrapping purposes.

The client installation space is located in /usr/local/share/fai. This directory contains directories class, disk_config, files, hooks, package_config, and scripts which contain the configuration files used during the installation process. The class directory contains the initial scripts for detecting hardware, partitioning hard drives, and defining classes that are read in alphabetical order. Defining classes during the installation process is simply anything that gets sent to standard output during the execution of these scripts. Also included in this directory are *.var files which set various parameters during the install. Files that contain hard disk configuration parameters such as partition sizes, formatting, and mount options are stored in the directory disk_config. The files directory contains files that will be copied to the client computers. Additionally, files contains a sub directory, packages, which houses a repository of custom packages that need to be included in the installation. The hooks directory contains user defined programs. Finally, package_config and scripts contain software packages from the distribution that will be installed and local customization scripts, respectively. Examples of these configuration space directories are included with the fai package. These files are quite generic and should work for testing purposes.

After the server and client configurations are complete, executing the command fai-setup will create the nfsroot directory and load the client configuration options. Fai-setup will produce many error messages during configuration, but all can be ignored for the most part. As long as “make-fai-nfsroot finished” and “Fai setup finished” are observed the setup process should be fine. The last step is to make bootable media for the clients.

For this particular installation, the client computers needed fixed ip addresses so a bootfloppy was prepared. The command make-fai-bootfloppy creates a bootfloppy with the proper configuration for a fixed ip address install. Actually make-fai-bootfloppy makes a bootable disk for different boot methods such as dhcp, bootp, fixed-ip, and rarp. Since we will be using the fixed ip method the actual network parameters need to be set by sending the network parameters to the kernel in the following format: ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<auto>. These parameters must be used as command line arguments to the make-fai-bootfloppy command to properly configure bootable media for installing a fixed ip address client.

The following command configures the bootable floppy: make-fai-bootfloppy -g -df "ip=(the above ip parameters) FAI_ACTION=install FAI_FLAGS=verbose, createvt, sshd" where flags g=grub bootloader, -df=use fixed address as default boot image, ip=kernel ip parameters (as above), FAI_ACTION=install=client installation, and FAI_FLAGS=verbose, createvt, ssh=verbose install, create virtual terminals, and start sshd for communication with the install server. The clients, when booted from this floppy, will receive all needed information for a fully automated installation.

SECURITY CONFIGURATION

Physical and Bios

A large number of installations will be required for public computing laboratories. Therefore, bios and bootloader protection will be necessary. The bios (basic input output system), a chip on the motherboard, is the interface between the computers hardware and the operating system. The bios configuration settings are saved to the cmos (complementary metal-oxide semiconductor), which is also a chip on the motherboard. Since this cmos chip requires a small amount of power to retain the bios settings, a battery on the motherboard is required. Removal of the cmos battery will cause the bios settings to be lost, thus, in effect bypassing any bios security settings.

In light of this problem, each computer case must be locked to prevent resetting the bios configuration. Each computer will have a lock inserted into the security tabs included on the computer case. The bios will also be set to prohibit booting from removable media such as cdrom or floppy disks. Unless restricted, removable media could be used to bypass the security configuration of the operating system. Therefore, the bios must be password protected to prevent unauthorized modification of the bios settings.

Bootloader

BIOS/CMOS security must be combined with protection of the bootloader to secure the booting process. Upon booting, the BIOS reads the master boot record and loads bootstrapping program known as the bootloader (Brouwer, 2001). The bootloader then transfers control of the computer to the operating system. Protection of this process is also essential to boot security.

Linux uses predominantly two bootloaders: lilo (LInux LOader) and grub (GRand Unified Bootloader). Both of these bootloaders were available for the installations. Though grub probably is the better bootloader, lilo was also needed for backward compatibility issues.   While the same security issues are present in both lilo and grub, the steps involved in securing each bootloader is somewhat different.

When lilo is chosen for the bootloader, upon execution, a lilo prompt is displayed on the computer screen. At this point certain keystrokes can halt the default boot process and allow any user to pass arguments to the kernel. In some Linux distributions typing "linux single" at the prompt boots the default kernel into single user mode and will produce a command line prompt with root permissions without ever requiring the entry of a password. Debian mitigates this problem somewhat by inserting the line

“~~:S:wait:/sbin/sulogin” into the file /etc/inittab. This command requires the entry of the root password before allowing access to single user mode. Unfortunately, issuing the command "linux rw init=/bin/bash" will bypass even this security measure and deposit the user at a single user prompt with root privileges, no password required (Hatch, 2002). To prevent these types of exploits security options can be set in /etc/lilo.conf. Setting the option password= [some password] will require that a password be entered every time the system is booted. This is probably not quite what we want due to the large number of computers involved as it would require the presence of a system administrator at each of the computers for every reboot. This problem can be solved by using the option restricted. Restricted only requires a password for booting when an option is passed to the kernel, normal booting will not require a password, exactly what is desired. The password chosen for the password command will be stored in /etc/lilo.conf in clear text so it is essential that it is only readable by root (chmod 600). After the configuration is set the command "lilo" must be issued to update the bootloader. One other important point that is not covered by much (or any) of the lilo documentation is after setting the password option and issuing the lilo command, the lilo password shows up in the /boot/map file in clear text. Therefore, this file should only be readable by root (chmod 600).

Grub exhibits essentially the same security problems as lilo. When the grub bootloader is executed, the user is presented with a grub prompt or a menu of bootable kernels listed in the /boot/menu.lst. If the menu.lst file is not password protected, a user has access to the actual boot commands. By highlighting a kernel in the boot menu and entering a few keystrokes the previously mentioned kernel arguments can be entered, thus effectively bypassing the operating system security. Basically, both bootloaders exhibit the same security weaknesses resulting from unprotected configuration files. Adding the option password --md5 [some password] to the menu.lst file will require entry of the grub password to modify boot commands. As an added security measure grub includes a utility grub-md5-crypt that will accept entry of a plaintext password and output its md5 hash. This hash can then be pasted into the menu.lst file after password --md5 thus eliminating the need to store a plaintext password (Free Software Foundation, 2004). This is marginally more secure than lilo but if access is gained to the menu.lst file the hash can be copied and entered into a password cracking program. Therefore it is still essential the menu.lst file be only readable by root with a recommended file mode of 600.

Operating System Security
With the BIOS/CMOS and bootloader security measures in place, the actual operating system security needed consideration. The following discussion will include local (non-network related issues) and network related security measures including communication protocols.

Filesystem
The installation divided the hard drive into various partitions and formatted the partitions using the ext3 filesystem. For all practical purposes partitions act as separate hard drives and are mounted individually during the boot process. While Linux can be

installed to a single partition, separation of the disk into numerous partitions allows the specification of mount options for each partition. Since each partition serves different purposes, mount options can enhance security by limiting functionality according to each partition's need. Partitions included in the install were /, /boot, /home, /opt, /usr, /var, and /tmp. Directories /opt and /usr contain mostly software, / contains software, device, and configuration files, /boot contains bootloaders and startup scripts, /home contains user directories, /var contains most log files, and /tmp contains temporary files. Numerous mount options exist in Linux but most distributions use defaults as the mount options. The default mount options are rw (read-write), suid (set-user-identifier allowed), dev (allow character or block devices), exec (allow binary execution), auto (auto detect filesystem), nouser (ordinary users cannot mount file system), and async (asynchronous io). More detailed explanations of each option can be found in the mount manpage. Mount options suid, exec, and dev along with their opposite's nosuid, noexec, and nodev are most important for securing the file system. The options nosuid, noexec, and nodev obviously prohibit suid, exec, and dev on the partition. Only the root partition should have the dev option enabled because it is the only partition where device files should exist. Also /tmp, /var, /home, and /opt should have the nosuid option enabled because there is no need to execute suid programs from these partitions. Following is an illustration of how mount options can help foil attackers. Since many scripting exploits install and execute files in the /tmp directory, using the mount option noexec will minimize the danger of these types of attacks. If the operating system was installed to a single partition, this type of security enhancement would not be possible.

Passwords

The debian package manager dpkg or if you prefer apt-get, requires user input for the installation of various software packages. Since our installations were automated with fai, input during the installs was not possible. Therefore, packages that need user input are installed with the default dpkg or apt-get configuration. Surprisingly, the default installation for the package passwd does not enable shadow passwords. This configuration would have terrible security consequences. When shadow passwords are not enabled, the encrypted passwords are stored in the /etc/passwd file which is world readable. An attacker would simply have to make a copy of this file and use a password cracking program like john the ripper to crack the passwords. Enabling shadow passwords replaces the passwords in /etc/passwd with an x and stores the encrypted passwords in the file /etc/shadow which is only readable by root (Fenzi, et. al., 2004). Therefore unauthorized users cannot access the encrypted passwords. When authentication is needed the program goes to the /etc/passwd file finds an x and then looks in the /etc/shadow file for the password. Due to the quite large security implications of shadow less passwords, they were turned on manually during the install by issuing the command “shadowconfig on”. Debian sets password policy in two files /etc/login.defs and /etc/pam.d/common/password. Maximum password length for a default install is set to 8 in /etc/login.defs. A maximum password length of 8 would not utilize the greater password lengths available when using an md5 encryption algorithm. Since older debian versions still use the crypt algorithm, this setting seems to be some kind of legacy setting. It actually has no effect on password length when md5 is the chosen encryption algorithm, but its presence causes confusion. Minimum password length is set at 4 which also seems quite short. For these installations minimum password length was increased to 6.

Inetd.conf
Since the main purpose of these installations were student and or faculty workstations, the practice of minimizing running services was observed. Inetd is a daemon that manages many types of incoming network connections. Incoming connections for each service controlled by the inetd daemon is controlled by its corresponding entry in the /etc/inetd.conf file. Entries for each service in the inetd.conf file consist of a single line containing: service (name assigned by /etc/services file), socket-type (usually stream (tcp) or dgram (udp)), protocol (usually tcp or udp), wait or nowait (only udp), user (user the process runs under), server (path to the server executable), and cmdline (command line options passed to the server). Traditionally, insecure daemons such telnet, ftp, rsh, rexec, and rlogin were managed by the inetd daemon. Debian provides additional security to the services managed by inetd by tricking the program to run tcp wrappers before the actual server program. In this way access to the server can be restricted and monitored. Tcp wrappers will be discussed in a following section. Although tcp wrappers restricts access to these services, they remain a large security liability because all passwords sent between the two computers are sent over the network in clear text. Since this practice has such large security implications because the passwords could be sniffed off the network, none of the aforementioned daemons were installed. Additionally, the internal services echo, discard, daytime, and chargen were all disabled because they are only used for debugging and testing purposes. The service time was also disabled because ntp will be used to synchronize system times. Rstatd (collection of kernel performance statistics) and rusersd (lists network users) were likewise disabled. Simply commenting out the appropriate line in /etc/inetd.conf and issuing the command kill –HUP pid of inetd disables the service. Debian has a somewhat more elegant utility that will comment out the appropriate line in the inetd.conf file and then restart inetd. For example, disabling rstatd and then restarting the daemon with the updated configuration is achieved by the single command update-inetd –disable rstatd (Pena, et. al., 2002). After final configuration, all services in /etc/inetd.conf were disabled.

Openssh
Openssh was installed as a secure replacement for telnet, ftp, rsh, rexec, and rlogin. As mentioned previously, communications with telnet, ftp, or the r-services transmitted unencrypted passwords and data over the network and thus were vulnerable to network sniffing, Openssh, encrypts all communications between client and server computers which essentially eliminates password sniffing. Since this is a client – server type of communication each computer needed to be both a client and server so data could be exchanged in either direction. Configuration options for the sshd daemon are stored in the file /etc/ssh/sshd_config. Some default configuration options that were changed to enhance security were PermitRootLogin and default protocol. The PermitRootLogin parameter determines if root logins are allowed through openssh. The default setting allows root logins through ssh. Permitting root to login through openssh make it virtually impossible to determine who is accessing the computer as root. This becomes

especially confusing if several people have root access. Therefore in these installs PermitRootLogin will be turned off so these logging ambiguities do not arise. Root access can still be granted by sshing into the computer with a users account and then suing to obtain root permissions. This behavior affords system administrators a better log trail of exactly who has been accessing the root account. Additionally the default setting for configuration option Protocol is 2,1 which means that ssh will use ssh protocol 2 to connect to the server if it is available and if not it will use protocol 1. This setting was changed to use ssh protocol 2, protocol 1 will not be used under any circumstances. Protocol 1 was not listed as an option because it contains many security vulnerabilities.

Some other configuration options which were not changed from the debian default values but merit some discussion due to their security implications were UsePrivilegeSeparation, RhostsAuthentication, and IgnoreRhosts (Barrett, et. al., 2001). The default value for UsePrivilegeSeparation was set to yes. After authentication, this option allows ssh to spawn a child process with the authenticated users permissions to handle the network connection. In this way if some vulnerability is exploited it will only have the permissions of the authenticated user and not root. RhostsAuthentication and IgnoreRhosts prevent .rhosts authentication and reading users .rhosts and .shosts files, respectively.

Tcpwrappers
Although tcpwrappers were previously mentioned as an access control mechanism for server programs managed by the inetd daemon, discussion about the actual program was postponed to the present section. Tcpwrappers are actually a server program, /usr/sbin/tcpd that is executed in place of the desired server daemon that provides logging and access control capabilities. If the connection passes the authentication tests the request is passed to the target server program. . It can be used by programs managed by inetd or standalone daemons such as ssh which are compiled against the libwrap library. Access and logging by the tcpd daemon is controlled by two files, /etc/hosts.allow and /etc/hosts.deny. When a network connection request is received by tcpd the hosts.allow file is checked first and if authorized the connection is allowed and tcpd forwards the request to the appropriate daemon. If none of the rules allow access in the hosts.allow file the hosts.deny file is queried. If the hosts.deny file denies the connection, it is refused (Frisch, 1995). Both of these files use the same syntax which is: <daemon_list> : <Client_list>: [<option>:<option>..]. The configuration of the hosts.deny file for the installations was ALL: ALL : DENY which means that all daemon access is refused to all hosts. Therefore, if the connection is not explicitly allowed in the hosts.allow file access is denied. The last entry in the hosts.deny file, DENY, is not strictly needed but makes it easier to quickly determine the rules purpose. The hosts.allow file contained an entry that allowed unlimited access to localhost: ALL: 127.0.0.1: ALLOW. This entry is needed for some local programs to work properly. There will also be an entry for unlimited access for administrator machines: ALL: xxx.xxx.xxx.xxx: ALLOW. Finally there will be an entry for the sshd daemon: sshd: xxx.xxx. :spawn (/usr/bin/safe_finger –l @%h |/usr/bin/mail –s “ALLOWED %s from %c” admin@somewhere.com) & : ALLOW. This entry allows ssh access to the sshd

daemon from the university class B subnet. Additionally, it utilizes the logging capabilities of tcpwrappers. Explained briefly, it only allows ssh connections from the university subnets, not internet access. A small number of computers with internet accessible ssh daemons are available to users. If a person is outside the university subnets, one of these computers can to be connected to and then one can ssh to any of the installation computers. The minimal number of computers running world accessible ssh daemons make it easier to keep track of potentially hostile entry attempts. Upon connecting to the computer the tcpwrapper program safe_finger, a safe form of finger, runs a reverse finger probe to the connecting computer to determine its address and sends mail to an administrator with the daemon accessed, the ip address of the server and the ip address of the connecting computer. Once the hosts.allow and hosts.deny files were configured they were checked with two other utilities provided with the tcpwrapper package. First, tcpdchk was executed to check the syntax of each file: tcpdchk -v which sends output to stdout. Finally tcpdmatch was executed: tcpdmatch ssh xxx.xxx.xxx.xxx which prints out if the connection is allowed or denied according to the rules in each file (Anonymous, 2000).

Sysctl and Iptables

The Linux command sysctl allows the modification of kernel runtime parameters stored in the /proc/sys directory. These parameters can be set in the /etc/sysctl.conf file which will be applied at bootup or they can be set manually. In many cases, for purposes of this discussion, the settings are just entering a 0 (disable) or 1 (enable) into the appropriate file. Networking parameters with security implications are located in the net directory. Some of the parameters set due to security concerns were icmp_echo_ignore_broadcasts, accept_source_route, tcp_syncookies, accept_redirects, send_redirects, rp_filter, and log_martians. Icmp_echo_ignore_ broadcasts were disabled so icmp echo requests were ignored to broadcast/multicast addresses. Source routed packets are packets that specify the exact route taken to a host which could potentially bypass security, therefore, accept_source_route was disabled. Disabling tcp_syncookies protects against flooding the syn backlog queue (Lechnyr, 2002). Since these computers do not act as routers accept_redirects and send_redirects were disabled. Rp_filter was enabled to prevent access from spoofed addresses and log_martians was enabled so packets with impossible addresses would be logged. More information can be found in the kernel source tree in the documentation directory, more precisely, in the file networking/ipsysctl.txt.

Netfilter/Iptables is the packet filter (firewall) for Linux kernels 2.4.x and 2.6.x. It replaces the older packet filter ipchains from the 2.2.x series kernels. Iptables main advantage over ipchains is in its connection tracking capabilities (stateful packet filtering). Stateful packet filtering is the ability to maintain state information in memory, such as source and destination ip address, port number pairs, protocol types, connection state and timeouts (Stephens, 2004). This property is very useful when writing packet filtering rules because it greatly reduces the number of rules needed for each type of connection. Ipchains, on the other hand, had no connection tracking abilities and writing packet filtering rules was quite a complicated process, especially if the firewalls default rules were to deny all traffic. Each client service had to have both incoming and outgoing rules for the connection to take place. In contrast iptables only requires outbound rules due to its connection tracking abilities; inbound traffic related to the outbound connection is allowed automatically.

Iptables allows the filtering of incoming, outgoing, and forwarded connections to the local computer by adding rules to the INPUT, OUTPUT, and FORWARD chains. The firewall configuration will use a default deny policy. Since none of these computers perform NAT (network address translation) for other machines the FORWARD chain will not be discussed further other than it is set to DENY by the default policy. A default deny policy is very restrictive, difficult to configure, and can cause many problems if it is not tested thoroughly but provides maximum protection. It is important for the script to include commands to load the desired kernel modules. Of particular importance is the module ip_conntrack_ftp which allows connection tracking for an ftp client. During an ftp session a control channel is opened between the client and the ftp server. When commands are issued another channel is opened to carry the data. If the ip_conntrack_ftp module is not loaded the firewall will not know what port this data is destined for and refuse the connection, thus rendering ftp unusable. Therefore the command modprobe ip_conntrack_ftp should be explicitly issued within the firewall script. Modprobe should be used instead of insmod because modprobe will load ip_conntrack_ftp and any other modules that it depends on, insmod will only load the requested module and if there are dependency conflicts the module will not load. While there are many internet sites with sample iptables scripts, (Ziegler, 2004) has example scripts that are easily understandable. Although many of these sites are a good starting point for packet filtering ideas, more complicated networking environments will require writing rules for services not found in any of these scripts. As an example, this university uses the AFS (Andrew File System) and custom rules had to be developed to access the file system. These rules are listed in Appendix 1.

Logging
Selected computers on each subnet had their log files sent to a logging server. Not all computers will send their log files to the server because this would create huge log files that would take an inordinate amount of time to check. Experience has shown that if these files become so large that it requires a considerable time investment to read them they will not get read. Of course if these files do not get analyzed there is no reason to keep them. Logging a few computers per subnet should be sufficient to pick up activity that otherwise might not be noticed. For example, a single computers log file will show that someone has scanned that particular machine and it might be missed or attributed to innocent error. On the other hand, since the logging host combines logs from numerous computers on different subnets it will be easy to see if scans are a systematic scan of the network. Logchecker was installed on the logging server to cull suspicious events from the log files and send them in an email to an administrator every hour. This greatly reduces the lines of log files that need to be read daily.

Security Updates
One of the major strengths of the debian Linux distribution is the ease with which it can be kept up-to-date. The package apt-get handles the installation and updates required

for each system. It is especially nice because it not only downloads and installs the desired packages but also solves any dependencies that might arise during this task. Apt-get can be configured to use many different types of file acquisition methods such as nfs mounted directories, cdrom, ftp, or http. Configuration options are located in /etc/apt/sources.list which direct apt-get where to look for the needed updates. It is important that /etc/apt/sources.list contains an entry for security updates so any newly discovered security vulnerabilities can be updated. Simply issuing the commands: 'Apt-get update' followed by 'apt-get upgrade' will keep each system up-to-date. All security updates are applied to a test computer first to see that no problems arise and if not all machines are then updated.

CONCLUSION

The proposed automated installation fulfilled the college's research computing needs and proved quite cost effective. Since debian is a free distribution, money could be spent on more high performance hardware. Installation and maintenance overhead for administrators remain quite low compared to many other operating systems. The security practices described in this paper have eased the security concerns about Linux in the college which is in stark contract to previous efforts to deploy Linux. Some minor problems have arisen which are directly attributable to the very restrictive firewall put in place on each computer that did not allow legitimate connections, but additional rules to the firewall script solved the problem. Overall the automatic installation and deployment have performed quite well.

REFERENCES

1) Anonymous. 2000. Maximum Linux Security. Sams Publishing. pp. 492-499.

2) Barrett, Daniel J. and Richard E. Silverman. 2001. SSH The Secure Shell, The Definitive Guide. O'Reilly & Associates. pp. 506-515.

3) Brouwer, Andries. 2001. "Large Disk HOWTO". URL: http://www.linuxdocs.org/HOWTOs/Large-Disk-HOWTO-5.html

4) Fenzi, Kevin and Dave Wreski. 2004. "Linux Security Howto". URL: http://tldp.org/HOWTO/Security-HOWTO/

5) Free Software Foundation. 2004. "GRUB manual". URL: http://www.gnu.org/software/grub/manual/grub.html#Security

6) Frisch, AEleen. 1995. Essential System Administration, 2nd Edition. O'reilly & Associates. pp. 625-626.

7) Hatch, Brian. 2002. "Another Backdoor to Root Access". URL: http://www.hackinglinuxexposed.com/articles/20020702.html

8) Lange, Thomas. 2004. "FAI Guide (Fully Automatic Installation)". URL: http://www.informatik.uni-koeln.de/fai/fai-guide.html/

9) Lechnyr, David. 2002. "Network Security with /proc/sys/net/ipv4". Linux Gazette:76. URL: http://www.linuxgazette.com/issue77/lechnyr.html

10) Pena, Javier Fernando-Sanguino, et al. 2002. "Securing Debian Manual". URL: http://www.linuxsecurity.com/docs/harden-doc/html/securing-debian-howto

11) Stephens, James C. 2004. Iptables. URL: http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html

12) Ziegler, Robert L. 2004. "Linux Firewall and Security Site". URL: http://www.linux-firewall-tools.com/linux

APPENDIX 1

Iptables rules to enable afs access

```
--snip
#AFS Authentication(udp)
for PROTOCOL in $PROTOCOLS; do
if [ "$CONNECTION_TRACKING" = "1" ]; then
                /sbin/iptables -A OUTPUT -o $INTERNET -p $PROTOCOL \
                        -s $IPADDR --sport $UNPRIVPORTS \
                        -d $LOCALNET3 --dport $AFSPORTS \
                        -m state --state NEW -j ACCEPT
fi

/sbin/iptables -A OUTPUT -o $INTERNET -p $PROTOCOL \
                -s $IPADDR --sport $UNPRIVPORTS \
                -d $LOCALNET3 --dport $AFSPORTS -j ACCEPT

/sbin/iptables -A INPUT -i $INTERNET -p $PROTOCOL \
                -s $LOCALNET3 --sport $AFSPORTS \
                -d $IPADDR --dport $UNPRIVPORTS -j ACCEPT
done
==============================================================
#AFS Minimum services
for PROTOCOL in $PROTOCOLS; do
if [ "$CONNECTION_TRACKING" = "1" ]; then
                /sbin/iptables -A OUTPUT -o $INTERNET -p $PROTOCOL \
                        -s $IPADDR --sport $AFSPORTS \
                        -d $LOCALNET3 --dport $AFSPORTS \
                        -m state --state NEW -j ACCEPT
fi

/sbin/iptables -A OUTPUT -o $INTERNET -p $PROTOCOL \
                -s $IPADDR --sport $AFSPORTS \
                -d $LOCALNET3 --dport $AFSPORTS -j ACCEPT

/sbin/iptables -A INPUT -i $INTERNET -p $PROTOCOL \
                -s $LOCALNET3 --sport $AFSPORTS \
                -d $IPADDR --dport $AFSPORTS -j ACCEPT
done
==============================================================
--snip
where: PROTOCOLS=”tcp udp”
       CONNECTION_TRACKING=”1” (connection tracking enabled)
       INTERNET=”eth0” (Ethernet card with internet access)
       IPADDR=”ip address of the local computer”
       AFSPORTS=”7000:7009”
```