Securing GNU/Linux

A Brief Introductory Guide (V. 1.1)

©2003, Andrew D. Hwang, ahwang@mathcs.holycross.edu

1 Introduction

GNU/Linux is a free (libre) clone of Unix, written from scratch by a worldwide group of volunteers. It is a powerful, flexible operating system increasingly used by individuals. Conversely, it is a sophisticated platform for launching attacks, and is sufficiently capable of networking to be exploitable remotely. This note is a brief guide to you, the new user/system administrator, who is charged with securing your machine against attack. The advice here is specifically geared toward RedHat and Mandrake, two of the most common distributions of GNU/Linux, but is generally applicable to all operating systems, and is (in particular) easily modified for Debian-based systems.

1.1 Terms of Usage

2 Network Services

The focus of this note is *network security*, which addresses the ways an intruder can break into and control your machine simply by virtue of your machine being connected to a network. If your machine is not networked, this note is of no practical use to you.

2.1 Clients and Servers

The *client-server model* is a common computer networking paradigm. The idea is that one entity, the *server*, provides a resource, such as access to a printer or monitor, web pages, a database, email forwarding, or login access. If you are accustomed to single-user operating systems (MacOS 9 or Windows 98, for example), you are probably not used to thinking of your computer as a server. However, even those operating systems allow file sharing, making the computer into a file-sharing server.

The other participant is the *client*, an entity that wishes to use a resource provided by a server. The following are typical client tasks: Sending a print job, displaying the output of a program that runs on another machine, browsing the world-wide web, sending email, or logging on to another machine.

In many organizations, ordinary users' machines are clients for most services, while servers run on powerful computers in locked rooms, maintained by a separate IT group. Because of this, users often refer to "the server" as if there were a single monolithic oracle to which ordinary users' requests are directed. The reality is not quite that simple: Each service (printing, email, web serving, login) has both a server and a client (or multiple clients, since usually several people use a single service). However, a server is not a computer, but a *program* running on some computer. Similarly, a client is a program; familiar examples include email readers, web browsers, file managers, and streaming media players. Note that the same machine may be a server and a client in multiple roles; a single machine may even be *both* server and client for some tasks, such as printing or displaying the output of graphical programs. If you have installed GNU/Linux, your machine is acting as a server for perhaps a dozen services, some of which are offered to the network by default; when you watch the messages that scroll up the screen when your machine starts up, you may see lines such as "`Starting bind`", "`Starting lpd`", and so forth. Each of these lines corresponds to a

service offered by your machine.

2.2 Networking Basics

Just as two people can communicate over long distance by telephone, two computers can communicate (over a physical wire, or by radio) with special hardware called a *network card*. The network card translates data internal to the computer into electrical signals that can be transmitted.

Machines nowadays communicate largely with the Internet Protocol (IP), a general-purpose, worldwide agreement on the meaning of the electrical signals transmitted between two network cards. This data is sent in the form of "packets", groups of up to about 1500 bytes that are something like pages of a postal letter.

The full story on networking is more complicated; hardware and IP are just two of seven network layers. For present purposes these layers suffice, but the transport layer (`TCP` and `UDP`) is of central importance in understanding the details of networking, and must be considered when creating a firewall.

Addressing

Every network card is manufactured with a unique MAC (media access control) address, rather like the serial number on an engine block. The MAC address is a globally unique identifier for a specific piece of network hardware.

Distant computers need not know each others' MAC addresses to communicate. Instead, every computer on the Internet has a unique *IP address*, 8 bytes (such as `192.133.83.197`, with each number between `0` and `255`) that function something like a postal address. By a system not unlike that used by the post office, two machines with IP addresses can communicate regardless of their physical location, and without there being a centralized "directory" of machines on the Internet. In addition, machines can be referred to by more human-readable names, like `www.google.com` or `radius.holycross.edu`. The "white pages lookup" that translates between these *fully-qualified domain names* and IP addresses is a service, called *name resolution* and provided by the *domain name system* (DNS).

As mentioned previously, a single computer may act as a server in several capacities. When a packet arrives at a machine, how does the machine know which server (program) should receive the packet's data? The answer is that a machine that communicates via IP has $65535 = 2^{16} - 1$ *ports* associated to

its IP address. These port numbers can be regarded as apartment numbers on the street address (IP address). When a client sends data to a server, the data is addressed to a specific port, whose number depends on the type of service.

For most services, there is general agreement on which port numbers are bound to which services. Under GNU/Linux, the file `/etc/services` contains a list of recognized services and their port numbers. A few common services are listed in Table 1. A browser connecting to the web server at `mathcs.holycross.edu` sends packets to `192.133.83.197:80`, for example.

Service	ftp	ssh	smtp	domain	http	https
Port	21	22	25	53	80	443

Table 1: A few common services and their port numbers.

2.3 Remote Attacks

In a benign world, a server would only be a liability because it consumes system resources (memory and processor time). In reality, a server is potentially a door by which an intruder can gain control of the machine running the server. Remember that a server is a program that functions normally by listening for requests that arrive from the network, then doing its best to comply, within the limitations of its ability. An ordinary request to a file server might ask for a file by name; the file server's job is to look for and return the appropriate file, or (if an error occurs) to generate an error message.

The danger is that a server must accept input from an untrusted source (the client), and it is not always the case that a server handles malformed requests safely and cleanly. One type of attack, based on a common programming mistake, is the *buffer overflow exploit*. For example, a programmer might write part of a file server to accept any filename that is at most 256 characters in length, which is more than enough for ordinary use. However, if more than 256 characters appear in the guise of a filename, the server must decide what to do with the extra data. If *all* the incoming data is written into the 256-byte box allocated by the programmer, it will be loaded into memory somewhere that the program does not expect. What happens next depends on the server binary, the operating system, and the skill of

the attacker who forged the bogus filename. Most likely the file server will only crash, causing a *denial of service* (DoS). However, a skilled attacker can sometimes force the server to run a useful command, such as starting a shell (command prompt) on a specified port. Since the file server is likely to be running as `root`, the login shell created by the attack will have `root` privileges. The attacker, who may be anywhere in the world, now has total remote control of the server machine.

While creating a buffer overflow exploit against a specific server requires considerable skill, running one requires none at all. What usually happens in the open source world is that someone discovers an exploitable flaw in a particular server, fixes the bug, and publishes both the vulnerability and the fix after notifying the vendor (if appropriate). The vendor publishes a security advisory, which system administrators (such as yourself) are expected to read, and to act upon by updating their software. Unfortunately, unpatched machines often remain on line for months or years after a vulnerability is published. Meanwhile, within a short time—sometimes within days—a skilled programmer (properly called a "cracker", not a "hacker") releases source code for an attack against the vulnerability, earning prestige in the system cracker community. At that point, legions of unskilled but maladjusted youths (properly called "vandals") download and launch the attack code, wreaking havoc on unpatched systems and bragging about their expertise. Security experts call these people "script kiddies".

Sometimes a cracker will write not merely an attack that can be launched manually, but a *worm*, a self-propagating program that spreads to vulnerable systems. A typical life cycle is this: The attacker starts the worm running on a compromised machine. The worm scans the Internet, looking for machines running the vulnerable server. When a remote attack succeeds, the newly-compromised machine is forced to download and compile the worm's source code, then to launch a new copy of the worm. A well-written worm will also clean up the compromised system so that it is extremely difficult to detect that an attack took place. A large organization may discover an attack only because a machine is responding slowly, or because it begins to attack other machines in the company's network.

3 Securing GNU/Linux

As the owner/administrator of a shiny new GNU/Linux box, you are probably sitting on a system cracker's gold mine. Common distributions install and enable numerous services by default, and some of them are notoriously vulnerable to remote exploits. Five of the most commonly implicated services are `bind` (DNS), `ftp` (file transfer protocol), `lpd` (line printer daemon), `rpc_statd` (remote procedure call, for the Network File System, NFS), and `sendmail` (email). If you are connecting your GNU/Linux box to the Internet, it is essential that you secure your machine against attacks.

It is a cliché (and a truism) that security is a frame of mind, not a do-once procedure or a program that you can run. There are habits to develop, but a few concrete things can be done immediately after installation, before a machine is connected to the Internet.

Commands below are shown with a root prompt, "`root#` ". Do not type the prompt when running a command.

3.1 Disabling Services

When GNU/Linux starts up, the system is initialized with shell scripts that lie in the directory `/etc/rc.d/init.d` (RedHat-based) or `/etc/init.d` (Debian-based). The instructions here assume a RedHat-based system.

The scripts mentioned above are not run directly, but through symbolic links. These symlinks are found in directories named `/etc/rc.d/rc[i].d`, in which `[i]` stands for an integer between 0 and 6, called the *runlevel*. The file `/etc/inittab` contains information about runlevels. Most people start in runlevel 5, which in RedHat presents a graphical login screen.

A symlink in a runlevel configuration directory is of the form `S10network` or `K09dm`, in which "S" stands for "start", "K" stands for "kill" (or stop), and the number determines the order in which the script is run. The remainder of the symlink's name is the name of the service started or stopped when the system enters or leaves the corresponding runlevel. The directory entries in Table 2 would start the network, then the system logger, the general-purpose mouse daemon (which enables highlighting, cut-and-pasting, etc.), the random number generator (used for encryption), and so forth. Non-standard user-specified configuration options are often put into a script called `rc.local`, which is run last.

10network	S20random	S55xntpd	S75keytable
S12syslog	S20xfs	S56xinetd	S90crond
S15gpm	S30dm	S55sshd	S99local

Table 2: Typical system configuration directory entries.

Most of the initialization scripts accept the four options `start`, `stop`, `status`, and `restart`. To stop `sendmail`, do

```
root# /sbin/service sendmail stop
```

As always when running commands as `root`, it is prudent (though not necessary) to type the complete path. If your distribution does not have the `service` command, the script can be run manually:

```
root# /etc/rc.d/init.d/sendmail stop
```

Stopping a running service does not ensure that the service will not be started at the next reboot. For that, you must remove the symlinks corresponding to services that you do not need. These symlinks could be managed by hand, but it is far easier (and less error-prone) to use the `chkconfig` utility. Examination of the actual initialization scripts should reveal a line such as "`chkconfig: 2345 60 60`" near the top of the script file. This line signifies that the script (say `cups`, the Common Unix Printing System) should be started and stopped in runlevels 2–5 inclusive, at order 60. To remove the symlinks that start `cups` automatically, do

```
root# /sbin/chkconfig --del cups
```

Use the `--add` option to add the relevant symlinks.

Removing the symlinks for a service ensures that the service is not started automatically at the next reboot. It does *not* stop the service if it is already running. If you are systematically disabling services, first stop the service, then remove the symlinks.

There has been a recent tendency for desktop managers (Gnome and KDE) to run numerous services. Be sure not to disable these by mistake or your desktop session will start to misbehave. You should be able to determine what an initialization script does by reading the script; if that fails, try looking for a manual page.

3.2 Firewalls

Services that must run can be protected from attack with a *firewall*, which is a program that examines incoming IP packets and accepts or rejects them according to criteria such as the port they are addressed to, the address they came from, whether or not they are part of an established connection, and so forth. Linux provides firewalling capabilities through `iptables` (`2.4` and later kernels) or `ipchains` (`2.2` kernel), which must be built into the kernel. Stock kernels have firewalling capability, but you must write and activate a *rule set*. The specifics of `iptables` are not covered here; instead, this note concentrates on the structure of a typical firewall, and the sorts of decisions you must make to write good firewall rule sets.

There are two general styles of granting permissions in computer security: "Everything not expressly forbidden is permitted", and "Everything not expressly permitted is forbidden". Note carefully that these are very different policies, because of the way they handle contingencies that have not been foreseen. When writing a firewall rule set, it is usually best to forbid everything not expressly permitted.

The Linux kernel's firewall code uses the concept of "chains", each of which is a gauntlet that a packet must traverse. The most obvious chain is the `INPUT` chain, which handles incoming IP packets. Like all firewall chains, the `INPUT` chain has a *default policy*, such as `ACCEPT` or `DENY`. If a packet successfully runs the gauntlet, it will have the default policy applied to it. As mentioned above, the default policy of the `INPUT` chain should be `DENY`.

In the Linux firewall code, packets are tested against the rules in a chain successively, and the first match wins. Firewall rules should be used to `DENY` access to ports on which services like `lpd` are running, unless you want the whole world to have access to those services (your printer, in this case). Even if you leave a port exposed, you can restrict the IP addresses that have access; for instance, you might run an `ssh` (secure shell, an encrypted login program) server, but only allow login attempts from machines within the network run by your school or company.

A useful feature of firewalls is their ability to *log* events, such as unauthorized attempts to connect to a blocked port. Further details on configuring a firewall can be found in the "Firewall and Proxy Server HOWTO" document, available (along with dozens of other useful HOWTOs) at no cost from the Linux Documentation Project:

`http://www.tldp.org/`

Almost every aspect of system administration is discussed in detail by at least one HOWTO.

To get a feel for the amount of potentially malicious traffic, you need only configure your firewall to log all packets where the "`SYN` flag" is set; such packets represent attempts by another computer to initiate a connection to a service on your computer using `TCP` (transmission control protocol). Unless you are behind a firewall already, expect to be scanned several times a minute, regardless of the time of day. Not all such packets are signs of an attack, but many of them are the equivalent of a burglar casing a neighborhood, seeing which houses are easy to break into.

3.3 Passwords

A *password* is a token that a user supplies in order to be granted access to a service, usually login. A good password is not an actual word in any human language, is at least 8 characters in length, and contains letters, digits, and punctuation. There is an art to choosing passwords, since the password should not be so difficult to type or remember that it must be written down, but not so obvious that anyone could guess it. The reason to avoid words is that if someone is able to obtain a copy of your encrypted password file (not as difficult as it sounds), they will run a *dictionary attack*, successively encrypting dictionary words and looking for a match among your encrypted passwords. For this reason, foreign words do not make acceptable passwords. Other tricks, such as replacing "s" with "5" or "i" with "1" are only a small improvement over dictionary words, but are better than nothing.

It is generally a good idea not to allow `root` to log on remotely. Instead, log on as an ordinary user, then use `su` (substitute user) to become `root`. You should also avoid using `ftp` and `telnet` to connect to your machine; both programs send unencrypted passwords, making it trivial to steal them. By default, `ssh` uses 1024-bit encryption, which is currently considered strong enough for military and diplomatic use.

3.4 Security Lists

In addition to the measures outlined above, you should regularly visit security-related sites, at which vulnerabilities are announced weekly, grouped by vendor. Your vendor also releases vulnerabilities as they are discovered, along

with patched packages; find out where to locate these announcements on the web, and/or subscribe to your vendor's security mailing list.

4 Summary

It pays to be reasonably paranoid when security is an issue, but it is equally crucial not to be overwhelmed by the prospect. Administering a GNU/Linux machine is an art that requires continual study, and security is an ongoing process. Happily, a great deal can be done just once to secure your GNU/Linux system. Disabling unneeded (and possibly dangerous) services and setting up a firewall are excellent first steps toward securing a system, and should be done immediately after installation, *before* connecting the machine to the Internet.

4.1 Further Reading

The resources below go into far more depth about security-related issues.

Perhaps the biggest security-related site is

`http://www.linuxsecurity.com/`

Vendors' announcements of vulnerabilities are generally made here, as well as on vendors' web sites.

The Honeynet Project,

`http://project.honeynet.org/`

is devoted to the study of system and network vulnerabilities "in the wild". Lance Spitzner has a number of excellent, detailed white papers on security that can be found here.

Dave Dittrich has written articles on "rootkits", the software an attacker installs on a machine once a successful attack has been run. A machine that has been "rootkitted" will lie cunningly to you, the administrator, about it's having been compromised. His web site contains a wealth of high-quality information:

`http://staff.washington.edu/dittrich/`