Security Evaluation of the Linux Operating System

Date: June 3, 2002
By: Craig L. Munsee and Chee Lee
Department of Electrical and Computer Engineering
Oregon State University, Corvallis, Oregon 97331 –USA.
e-mail: munsee@engr.orst.edu, leech@engr.orst.edu

Abstract

Linux is an open source operating system that has gained much popularity. More and more people are using it for a variety of tasks. However, due to its open source nature, how secure is Linux? And are these people just setting themselves up for an attack? These are viable questions that every Linux user should be aware of. Therefore, this report will examine the overall security of Linux as a server as well as provide some possible solutions for increasing security.

1 - Introduction

Since its birth in 1991, Linux has grown to become one of the world's most popular operating systems. Students like it for the price and the open source flexibility. Network administrators like it because it can communicate with many other operating systems and run on virtually any processor. Internet Service Providers (ISPs) like it because of the native Internet support that it provides. Even with all the strengths of Linux, many claim that Linux isn't secure because of its open source nature. Some feel that the open source code makes it easier for attackers to find and exploit flaws in the operating system. This paper will look at Linux as an open source operating system. We will look at the types of attacks that are used to gain access to a Linux network. We will also see how secure Linux is, compared to other commercial operating systems. We will conclude this paper with some recommendations on what can be done to make Linux more secure.

2 - Linux an Open Source Operating System

The GNU project (GNU is a recursive acronym for "GNU’s Not Unix") was started in 1984 by Richard Stallman, a researcher at MIT's Artificial Intelligence labs, in reaction to the (then) new practice of keeping source code secret and enforcing software licensing [8]. Stallman saw the withdrawal of source code as a curtailment of a programmer's freedom to modify and improve software [8]. He also saw the license restriction on copying as being at odds with his philosophy of being a good neighbor and sharing ideas. Stallman's goal was to recreate a complete operating environment that was free of such restrictions, with all the tools and utilities that a computer user would ever need [8]. He chose to model this new operating environment after the Unix operating system (OS).
Stallman realized that if he just made his code available to everyone, that it could easily be copied by someone else, who could modify it, copyright it, and not make the new code available to everyone. For this reason Stallman chose to copyright his material with the constraint that if any code was copied that the new modified code had to be made available to everyone. The conditions are that anyone modifying the code for later redistribution has to make their source code public on the same terms [8]. This copyright became known as the GNU General Public License.

Stallman wasn't successful in creating a completely new operating environment, but had created many peripheral utilities. In 1991, a Finnish Computer Science student named Linus Torvalds wrote the first version of a Unix-like kernel for his own use, and posted the code on the Internet with a request to other programmers to help him build it up into a working system [8]. He incorporated the work that Stallman produced and called this new operating system Linux. Torvalds released Linux under the GNU General Public License just like the GNU project.

3 - The Popularity of Linux

Several things have made Linux a popular operating system. Some of the strengths of Linux are listed below.

3.1 - Zero Price Tag

Other commercial operating systems require that a new license be purchased for each computer it is installed on. Due to the GNU General Public License, Linux does not have this same restriction. This can reduce the cost to a company with several computers by a significant amount. We also need to remember even though Linux is free, there wouldn't be any benefit if it didn't do the job.

3.2 - Flexibility

If you used a non-open source OS and needed something special for your company, you would have to ask the manufacturer to make this change for you. Most manufactures of a commercial OS are not interested in customizing the OS for each company. Even if they were, this would cost a lot of money. With Linux you have the source code that you can freely customize to your needs.

3.3 - Stability

Unix is known for its stability. This is one of the reasons that Linux was modeled after it. Linux has the advantage of a quarter century of Unix experience to draw on. Most significantly, the open-source code model of Linux seems to ensure that bugs are detected and fixed early [8].
3.4 - Compliance

Due to the GNU General Public License, Linux cannot have proprietary features. The license therefore ensures that the only changes to the system that will last are those that are accepted by the "community" [8]. The community has no vested interest in creating proprietary standards and protocols, and so the OS naturally coalesces around industry standards [8]. Linux is a POSIX-compliant (Standards Defining Unix) OS and it supports ANSI, ISO, IETF and W3C standards.

3.5 - Hardware Support

Linux will run on virtually every known processor, whether RISC or CISC, 32-bit or 64-bit. The most common processor for Linux is of course, the Intel x86 family, but it also runs on Motorola's 68k, the IBM/Apple/Motorola PowerPC, Compaq/Digital's Alpha, MIPS chips, Sun's SPARC and UltraSparc and Intel's StrongARM [8]. Intel has recognized the popularity of Linux and has made an objective to make Linux run fastest on its chips. Intel is pushing its Uniform Driver Interface (UDI) as a common Unix approach to device drivers, and is trying to get the Linux community to help write the drivers [8]. Another Linux strength is that Linux has the ability to run on older computers with less memory and disk capacity than other operating systems. The one draw back is that not all peripherals and cards are supported by Linux, but as the popularity for Linux grows this will change.

3.6 - Native Internet Support

Open Source Apache, the world’s most popular webserver, runs naturally on Linux. Add-on modules like mod_perl allows Perl CGI scripts to be interpreted and run within Apache’s memory space. The mod_jserv module allows Apache to use Java servlets [8]. The mod_php module allows Apache to run HTML-embedded scripts in a Perl-like Hypertext Pre-Processor language called PHP, a program that works exactly analogously to Microsoft’s Active Server Pages [8]. Linux also supports firewalls like ipchains. These things and others have made Linux a very popular server OS among Internet Service Providers (ISPs). Linux is an excellent, standard platform for web applications. You can use it to build a complete, secure Internet site, including router, firewall, proxy, webserver, mailserver, database server and directory server [8].

3.7 - Interoperability with Existing Systems

As a server Linux needs to be able to coexist with other operating systems. Linux can talk SPX/IPX in a Netware environment, Appletalk in a Mac crowd, and even SNA to IBM mainframes [8]. Linux can even coexist with Windows systems because it is able to speak TCP/IP. One of the newest languages that Linux can speak is Session Message Block (SMB) protocol. SMB is the protocol that Windows9x/2000 operating systems use for sharing files and printers.
4 - Types of Attacks

In order to evaluate the security of Linux we need to look at the attacks that are being used against it. We do not intend to look at every attack that has been used against Linux, but will discuss some of the more common ones. For a complete list of all security vulnerabilities see http://www.linuxsecurity.com. Attackers have several ways to attack a server. The utilities to carry out these attacks are readily available on the Internet to anyone who wants to find them. These attacks can range from a simple test to see what version of OS you are running to having complete control of your system. The attacks can come from an outside source like a hacker on the Internet, or an inside source like an employee at a workstation. Some of the different types of attacks that an attacker can use will be discussed in greater detail below.

4.1 - Scanning

The first thing an attacker does is find out who they can attack. They can't attack you if they don't know you even exist. Attackers typically use what is called War Dialing to find networks to attack. Once they know that you exist they can use other tools to find out what type and version of OS you are running on your system. An attacker can employ what is called a packet sniffer. The sniffer listens on an Ethernet port for things like passwd, login, and su. When these things are detected the attacker can gain passwords that give him access to the network. Packet sniffers are deployed on an already compromised port or can be done on the inside from a laptop or another computer. Using ssh or other encrypted password methods thwarts this attack. Things like APOP for POP accounts also prevents this attack. (Normal POP logins are very vulnerable to this, as is anything that sends clear-text passwords over the network.) [10]

4.2 - Known Vulnerabilities and Misconfigured Services

If a network is not kept up to date with security patches it can easily be attacked. The latest vulnerabilities can be found at http://www.linuxsecurity.com by anyone. The attacker is hoping you have not been diligent in securing your network and will try to catch you with one of these vulnerabilities.

Another way an attacker can gain access to your network is through a service that has been left open or unconfigured. This leaves the services with the default settings and passwords. The default settings are known to an attacker and can be used to gain access to your network.

4.3 - Worm Attacks

Worms are programs that replicate themselves, but are not viruses. Worms don’t attach themselves to other programs or modify the code of other programs. Worms scan systems for exploitable holes like LPRng, rpc-statd, wu-ftp and BIND. Worms use these holes to gain root access and install a backdoor on a system. The worm will then mail the systems information to specific e-mail addresses to notify the author about the attack. The worm will then use this
system and its bandwidth to continue its attack on other systems. This allows the worm to grow at an exponential rate.

4.4 - Trojan Horse Programs

The Trojan Horse attack is known as one of the oldest tricks in the book, but can catch you if you aren't looking for it. A Trojan horse can be disguised as a useful program, utility, or a fun game. You can download it off the Internet, or a friend that doesn't know it's a Trojan horse could send it to you. When this program has been installed, the Trojan horse will not give any indication that you have been attacked. The Trojan horse can give anyone who connects with your computer, using the Trojan horse as a remote server, access to your hard drive. This doesn't just make you vulnerable to the creator of the Trojan horse, but to other attackers too. This attack can be used to find personal or financial information that can be exploited by the attacker. Trojan horse programs come in the form of binary only. Few Attackers are willing to release source code that can be publicly scrutinized. This is one of the strengths of the open source community surrounded by Linux.

4.5 - Direct Physical Access or Local Hacking

Local hacking can be defined as the way of gaining access to a computer while actually sitting at it, and is done using an input device such as a CD, floppy, or zip disk. In general, all computers are vulnerable to local hacking - some more than others. Local hacking, however, is not necessarily a bad practice. It can be beneficial in certain situations. For example, if the password to a computer with important information on it is lost, local hacking can provide a means for retrieving that information. Or if a computer contains potential evidence for a case but is password protected, local hacking can help authorities obtain that evidence. But just as local hacking can be used for a good purpose, it can also be used for a bad purpose such as gaining unauthorized access.

Linux, consequently, is even more vulnerable to local hacking than Windows NT or XP. In order to locally hack Windows, a specialized boot disk, such as ERD Commander, is needed. However, to locally hack Linux, all that is needed is a Linux installation disk that gives the user a command prompt. One such disk is Slackware's install disc. The reason Linux is so much more vulnerable is the fact that when a user logs in as root, the user has root access everywhere.

So using the installation disk, a hacker can boot up to the command prompt and log in as root on the installation disk. And even though the hacker does not know the actual system root password, the hacker can gain root access to the system since he or she is already logged in as root. All the hacker needs to do after he or she logs in on the installation disk is, instead of running the installation setup, just mount the Linux system partition by typing `mount -t ext2 /dev/hda1 /mnt` [1]. The hacker can mount this partition because the Linux OS treats his or her root access the same as the system root access.

With the system partition mounted, the hacker has access to all the important Linux files and can do nearly anything he or she wants. One of the most damaging is that the hacker can change all the commands that are executed from the installation disk to take their paths from the system partition by typing `chroot /mnt` [1]. Doing this allows the command `passwd`, which
changes the password for any user (if the executing user has a higher level), to access the system file `/etc/shadow` instead of the installation disk file `/etc/shadow`. The danger with this is that the system file `/etc/shadow` is the password file for all the users on that particular computer while the installation disk file `/etc/shadow` contains only the password to allow installation (usually empty). The hacker could change the system root password so that he or she could remotely access the system at a later date. Or a sly hacker could setup a user for him or herself with root privileges. The later is extremely dangerous because it could go undetected for quite a long time if the system administrator does not check the logs that often.

Unfortunately, there is no good solution to this vulnerability yet. The best course of action is to restrict or deter people from gaining local access to the Linux computer. This can be done by physically placing the computer in an area inaccessible to all but the necessary people or by monitoring who uses the computer with equipment such as a video camera. On another note, local hacking does leave a lot of traces. If a computer is hacked locally, the perpetrator could easily be tracked down and apprehended. This may also serve as a deterrent for local hacking. However, other methods also need to be considered.

4.6 - Spoofing

Spoofing is a way that an attacker can masquerade as someone else. The attacker's exploits take advantage of a trust-relationship that already exists. Spoofing of network connections involves forging an IP source address to trick the destination into thinking you are someone you really aren't [9]. Spoofing of network services involves using poorly configured (or misconfigured) applications, typically SMTP, to trick the client, server, or recipient into thinking you are someone you are not [9].

4.7 - Denial of Service Attacks

A Denial of Service (DoS) attack is where the attacker prevents users from using a service on the network. Denial of service attacks either try to make some resource too busy to answer legitimate service requests, or to deny legitimate users access to a machine [9]. A DoS attack can also be used to keep a victim busy while the attacker impersonates the host. This is known as a Man-in-the-middle Attack. The Man-in-the-middle Attack will be discussed in greater detail below.

4.7.1 - SSH Man-in-the-middle Attack

Even when all the security precautions are in place and the Linux system is secure from a direct attack, it is still possible to gain unauthorized access using Linux's `ssh` software and the man-in-the-middle attack. The man-in-the-middle attack simply refers to someone eavesdropping on the communication between two parties.

A hacker can use the Linux program `sshmitm` to perform a `ssh` man-in-the-middle attack on a `ssh` session. The machine running `sshmitm` needs to be situated between the `ssh` client and `ssh` server. When the client machine connects to the server, the machine running `sshmitm`
intercepts the packets. The sshmitm process pretends to be the actual server. It will follow the
SSH protocol specification, providing the host key, agreeing on cryptographic algorithms and
keys, and eventually asking the client for the password. The sshmitm program establishes a
connection to the actual server, as if it’s a normal ssh client. It takes the data that it receives from
the real client, which it has available in clear text, and re-encrypts it to the real ssh server as
appropriate. Because, in the end, the client sends packets to the server and everything looks
normal, there’s no way to realize anything is fishy once the connection is established. However
each and every byte in those packets is readable (and modifiable) by the sshmitm program. [4]

The man-in-the-middle attack can easily be prevented by educating users about the nature
of ssh. In order to establish a secure ssh connection, the ssh host key of the server needs to be
verified. The first time the client connects to the server, it is presented the ssh host key of the
server. During this first connection, there is no way - short of contacting the administrator - to
verify if the key presented is the correct fingerprint or not. But after the host key is accepted, it
is entered into ~/.ssh/known_hosts so that it can be compared with later sessions. This host key
is the vital component in preventing a ssh man-in-the-middle attack.

The sshmitm program does not have a copy of the actual ssh host key for the server it is
impersonating. Therefore, it cannot present the correct key to the client and must generate its
own fake host key. Since a fake host key is in use, the user connecting will see the following
message, "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED," which should
be an immediate red flag indicating that something is possibly wrong. However, most users
ignore the warning and continue on with their connection allowing sshmitm to do its dirty work.
The principle that most users will ignore the warning is what allows the ssh man-in-the-middle
attack to be effective. Thus, educated users who make it a habit to verify the host key is the best
solution to this attack.

4.8 - Program Code Exploits

Program code exploits are one of the largest security vulnerabilities that exist. Programs
are constantly being updated to remove these vulnerabilities, but not everyone keeps up with the
latest version of a program. This type of attack takes advantage of a common coding style, like
never allocating enough buffer space and never checking for an overflow. Other attacks of this
type can take advantage of directories, like /tmp, that are typically used for temporary files. This
makes it possible to write into an existing file or to be recovered after they have been deleted.
Buffer overflows and the Dangers of isof and proc are discussed in greater detail below.

4.8.1 - Buffer Overflows

Buffer overflows occur when a program attempts to write data into a memory location
that is smaller than the data. This can result in the program becoming corrupted causing it to fail.
Even worse, if an attacker carefully constructs the data that overflows, he or she can gain
unauthorized access to the system or even higher access privileges to the system.

Consider the following piece of code:
#include <stdio.h>

int main() {
    char buff[15] = {0};
    printf("Enter your name:");
    scanf(buff, ",");
}

This program reads a string from the standard input (usually the keyboard) but doesn’t check the string’s length. If the string has more than 14 characters, then it causes a buffer overflow because scanf() tries to write the extra characters past buff’s end. The result is usually a segmentation fault or core dump that crashes the program. And in certain situations, the user executing the code will receive a shell prompt after the program crashes. [5]

Normally, a shell prompt would not be a security issue because each shell has a privilege level and as long as the attacker does not know the root password, he or she will never have a shell with root privileges. However, in certain cases, a program needs to be able to run at a higher privilege level than the level of the process or user that initiated it. In a Linux system, programs that are marked SUID can run with higher privilege levels than the levels of their initiator. This is where the security problem arises.

If an attacker were able to cause a program with a higher privilege level than the user to crash and open a shell prompt, he or she would now have a shell with a higher privilege level, and be able to do what he or she would not normally be able to do. In fact, some programs have root privileges, which means that they have complete control of the computer system no matter the privilege level of the person who initiated the program. Attacking such programs through buffer overflow and causing them to open a shell is an easy method for attackers to gain root access to a computer system. [6]

Another dangerous situation can also occur when the program does not crash but instead continues execution. An attacker familiar with the system’s internal structure such as memory locations, memory addresses, etc. can create a string that is just long enough to overwrite the programs instruction pointer (IP) - a pointer to the program’s next instruction [5]. If the last couple of bytes of the string contain a valid memory address, then the program can be altered to execute code in that new memory location skipping the code that it was supposed to execute before its IP was overwritten. The new code could be a rogue program. Or the code that was skipped was supposed to perform security checks, so now the attacker can continue on without the fear of being detected.

Buffer overflows occur primarily due to failure to observer correct programming procedures:
- Failure to check boundary conditions.
- Failure to check return codes of functions and procedures.
- Pointer problems/wraparound. [7]

Many of the buffer overflow problems can be found and corrected before the software is released. All that needs to be done is audit the software for vulnerabilities and practice good programming techniques. However, since Linux is an open source OS with many different programmers, the programming techniques are bound to vary. Also, there is no centralized unit for auditing Linux software before it is released. Thus, Linux will be more susceptible to buffer overflows.
overflow problems. However, due to the large audience of supporters, Linux bugs tend to be caught and fixed faster than non-open source systems.

4.8.2 - Dangers of Isof and Proc

`isof` and `/proc` can be very handy when files are accidentally deleted. For example, let's assume that the movie file `my_wedding.avi` is open. And while it is still open, it is accidentally deleted—you typed `rm ski *.avi` instead of `ski*.avi` deleting all movie files instead of all ski movie files. The bad news is that `my_wedding.avi` can no longer be accessed using conventional methods. However, the good news is that since the movie file is still open, it can still be recovered—it cannot just be resaved because most movie players do not have the option to save an open movie. But using `isof` and `/proc`, the movie can be recovered because in Linux, files aren't actually removed from disk until all open file descriptors are closed and all hard links to the data are removed. [2]

`isof` lists all the open files and their file descriptors for a certain process—in this case, the movie player. From this list of open files, the `my_wedding.avi`'s file descriptor can be found. And using this file descriptor, the movie can be copied back onto the disk. The reason this method works is because the movie is still open and in memory under the `/proc` file system. The `/proc` file system is not an actual directory on disk like `/usr` or `/home`. Instead, `/proc` is a directory-based view of information the kernel makes available. When the copy is performed, the movie is actually being copied from memory back onto the disk. [2]

The capability of recovering a file can be exploited by savvy hackers to avoid being detected. A common trick is to open a file and then immediately delete it so that the file is not visible on the machine to tools like `find`, `locate`, etc. And since the file will be in memory, if the machine is rebooted, the file will disappear as well. Also, until the file is closed, it is still completely useable. So a malicious hacker could start a rogue program, delete it, and still have it run virtually undetectable. Even if the administrator determines that something is amiss and kills the process or reboots the machine, the incriminating files will disappear leaving little if no trace of the hacker at all.

Since `isof` and `/proc` are inherent capabilities of Linux, there are really no solutions to their exploitation. The only way to prevent `isof` and `/proc` from being exploited is to keep hackers away from the system through other means. However, if a hacker does run a rogue program by exploiting `isof` and `/proc`, the system administrator can use those same tools to locate the rogue program and make a copy of it as evidence. The administrator just has to keep in mind to use `isof` and `/proc` before rebooting the system. [3]

5 - Linux Vulnerabilities

The attacks that were discussed above are just some of the attacks that are used to attack a network. Linux is not the only OS that is susceptible to these attacks. In order to evaluate Linux as a server we need to compare it to other operating systems. We will use some information that was gathered by Security Focus Online [11] for this comparison. Security Focus recorded the number of security vulnerabilities that were found for each OS. Table 1 shows the Number of OS

### Number of OS Vulnerabilities by Year

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>21</td>
<td>38</td>
<td>10</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>BSD/OS</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>BeOS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Caldera</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>Connectiva</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Debian</td>
<td>3</td>
<td>2</td>
<td>31</td>
<td>55</td>
<td>28</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>5</td>
<td>2</td>
<td>17</td>
<td>36</td>
<td>17</td>
</tr>
<tr>
<td>HP-UX</td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>26</td>
<td>16</td>
</tr>
<tr>
<td>IRIX</td>
<td>28</td>
<td>15</td>
<td>9</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>MacOS</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>MacOS X Server</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mandrake</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>46</td>
<td>36</td>
</tr>
<tr>
<td>NetBSD</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>Netware</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>Red Hat</td>
<td>6</td>
<td>10</td>
<td>47</td>
<td>95</td>
<td>54</td>
</tr>
<tr>
<td>SCO Unix</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>Slackware</td>
<td>4</td>
<td>8</td>
<td>11</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>Solaris</td>
<td>24</td>
<td>33</td>
<td>34</td>
<td>22</td>
<td>33</td>
</tr>
<tr>
<td>SuSE</td>
<td>0</td>
<td>1</td>
<td>23</td>
<td>31</td>
<td>21</td>
</tr>
<tr>
<td>TurboLinux</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Unixware</td>
<td>2</td>
<td>3</td>
<td>14</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Windows 3.1x/95/98</td>
<td>3</td>
<td>1</td>
<td>46</td>
<td>40</td>
<td>14</td>
</tr>
<tr>
<td>Windows NT/2000</td>
<td>10</td>
<td>8</td>
<td>78</td>
<td>97</td>
<td>42</td>
</tr>
</tbody>
</table>

**Table 1**: Number of OS Vulnerabilities by Year [11].

### Top Vulnerable Packages 2000

<table>
<thead>
<tr>
<th>Packages</th>
<th># Vulns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows NT 4.0</td>
<td>71</td>
</tr>
<tr>
<td>RedHat Linux 6.2 i386</td>
<td>65</td>
</tr>
<tr>
<td>RedHat Linux 6.2 sparc</td>
<td>53</td>
</tr>
<tr>
<td>RedHat Linux 6.2 alpha</td>
<td>53</td>
</tr>
<tr>
<td>Microsoft Windows 2000</td>
<td>52</td>
</tr>
<tr>
<td>Debian Linux 2.2</td>
<td>48</td>
</tr>
<tr>
<td>RedHat Linux 6.1 i386</td>
<td>47</td>
</tr>
<tr>
<td>Microsoft Windows 98</td>
<td>40</td>
</tr>
<tr>
<td>RedHat Linux 6.1 sparc</td>
<td>39</td>
</tr>
<tr>
<td>RedHat Linux 6.1 alpha</td>
<td>39</td>
</tr>
<tr>
<td>MandrakeSoft Linux Mandrake 7.0</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Windows 95</td>
<td>35</td>
</tr>
<tr>
<td>RedHat Linux 6.0 i386</td>
<td>33</td>
</tr>
<tr>
<td>Microsoft IIS 4.0</td>
<td>29</td>
</tr>
<tr>
<td>Microsoft BackOffice 4.5</td>
<td>29</td>
</tr>
</tbody>
</table>
Microsoft BackOffice 4.0 29
RedHat Linux 7.0 28
MandrakeSoft Linux Mandrake 7.1 26
RedHat Linux 6.0 alpha 25
Conectiva Linux 5.1 25

Table 2: Top Vulnerable Packages for 2000 [11].

<table>
<thead>
<tr>
<th>Packages</th>
<th># Vulns</th>
</tr>
</thead>
<tbody>
<tr>
<td>MandrakeSoft Linux Mandrake 7.2</td>
<td>33</td>
</tr>
<tr>
<td>RedHat Linux 7.0</td>
<td>28</td>
</tr>
<tr>
<td>MandrakeSoft Linux Mandrake 7.1</td>
<td>27</td>
</tr>
<tr>
<td>Debian Linux 2.2</td>
<td>26</td>
</tr>
<tr>
<td>Sun Solaris 8.0</td>
<td>24</td>
</tr>
<tr>
<td>Sun Solaris 7.0</td>
<td>24</td>
</tr>
<tr>
<td>Microsoft Windows 2000</td>
<td>24</td>
</tr>
<tr>
<td>MandrakeSoft Linux Mandrake 7.0</td>
<td>22</td>
</tr>
<tr>
<td>SCO Open Server 5.0.6</td>
<td>21</td>
</tr>
<tr>
<td>RedHat Linux 6.2 i386</td>
<td>20</td>
</tr>
<tr>
<td>MandrakeSoft Linux Mandrake 6.1</td>
<td>20</td>
</tr>
<tr>
<td>MandrakeSoft Linux Mandrake 6.0</td>
<td>20</td>
</tr>
<tr>
<td>Wirex Immunix OS 7.0-Beta</td>
<td>19</td>
</tr>
<tr>
<td>Sun Solaris 2.6</td>
<td>19</td>
</tr>
<tr>
<td>RedHat Linux 6.2 sparc</td>
<td>18</td>
</tr>
<tr>
<td>RedHat Linux 6.2 alpha</td>
<td>18</td>
</tr>
<tr>
<td>Debian Linux 2.2 sparc</td>
<td>18</td>
</tr>
<tr>
<td>Debian Linux 2.2 arm</td>
<td>18</td>
</tr>
<tr>
<td>Debian Linux 2.2 alpha</td>
<td>18</td>
</tr>
<tr>
<td>Debian Linux 2.2 68k</td>
<td>18</td>
</tr>
</tbody>
</table>


Security Focus Online said this about the data collected. There is a distinct difference in the way that vulnerabilities are counted for Microsoft Windows and other operating systems [11]. For instance, applications for Linux and BSD are often grouped in as sub-components with the operating systems that they are shipped with [11]. For Windows, applications and sub-components such as Explorer often have their own packages that are considered vulnerable or not vulnerable outside of Windows and therefore may not be included in the count [11].

With this said, we will look at the tables and try to get an unbiased view of the vulnerabilities of the different operating systems. According to Table 1, we can see that almost all operating systems increased in the number of vulnerabilities for the year 2000, and for the year 2001 the number dropped slightly. This may indicate that as the different operating systems increase in complexity, so do the vulnerabilities.

From tables 2 and 3 we can see the operating systems that have the highest number of vulnerabilities. We note here that the different versions of Linux as well as Windows 2000, and Sun Solaris, are listed as some of the most vulnerable. The different versions of Linux seem to have a large number of vulnerabilities; this may be due to its open source code.
Attackers can easily search through the Linux code for vulnerabilities and trying to exploit them. Other systems that don't have open source are not as easily probed for flaws that can be exploited. Does this make the other systems more secure than Linux? The answer to this question is no. Just as easily as an attacker can sift through the Linux code to find weaknesses so can the network administrators to fix the problems. When it comes to commercial operating systems, the only way a vulnerability can be found is through an attack, and the only way a vulnerability can be fixed is through the manufacturer. Linux vulnerabilities are fixed almost as quickly as they are found. The Linux community works together, to fix problems as quickly as possible. As for other operating systems, unless the problem is major, the fix will have to wait.

Now for the big question, is Linux secure? The answer here is no, at least not out of the box, but neither are some of the other popular operating systems.

6 - Making Linux Secure

As we saw from above, Linux is not a secure OS coming strait out of the box. Here we will look at some of the things that need to be done, to make Linux a secure OS, or at least as secure as it can be.

Start with installing the patches that are found at, http://www.linuxsecurity.com to fix many of the known securities vulnerabilities. Also keep up to date with the other software you use. Make sure the security vulnerabilities that exist in these programs are also patched.

The next thing that needs to be done is configure Linux completely. If you don't need a service disable it. Ask questions like, does everyone on the Internet need to have access to the ftp server or have access to in.fingerd? Don't give more services than the bare minimum. This can be done by setting up permissions. Use tools like tcpd. tcpd is called into action from another daemon, inetd, whenever someone tries to access a service like in.telnetd, wu.ftpd, in.fingerd, inrshd, etc. tcpd's job is to look at two files and determine if the person who is trying to access the service has permission or not [12]. The two files that are checked are /etc/hosts.allow and /etc/hosts.deny. These files are empty by default making it possible for anyone to have access to the network. Take time to list those that do have access and make sure you list those that don't have access. The best would be to set the deny file to deny all, and list those that you want to have access in the allow file, since the allow file is checked first.

The next thing that should be done is making sure all users have good passwords. Below you will find six recommendations for good passwords. Using these recommendations will make it much more difficult for an attacker to obtain the passwords.

Password Recommendations:
1. Make the password 6 to 8 characters long.
2. Don't use words found in the dictionary.
3. Mix the case of the password.
4. Use at least one number.
5. Use at least one non-alphanumeric character.
6. Change the passwords on a regular basis.
The last thing that we will recommend here is to keep an eye on /var/adm/syslog and /var/adm/messages at least once a day. Watch for irregular activity. This will give you information about attack attempts.

7 - Conclusion

Throughout this paper we have evaluated the Linux operating system. We looked at how Linux was created and it's open source environment. We looked at many of the strengths that have made Linux popular today. Several of the attacks that can be used to gain access to a Linux network have been discussed. Using information that was obtained from Security Focus Online, we have compared the security of Linux to other commercial operating systems. We concluded that Linux, as well as other commercial operating systems, aren't secure straight out of the box. We have recommended some things that can be done to make Linux more secure.
References


Example Project Paper, December 1999,

[7] "Unix Flaws and Vulnerabilities," ECE 478/578 Example Project Paper,


http://www.linuxsecurity.com/docs/LDP/Security-HOWTO.html#toc8, accessed May 31,
2002.
