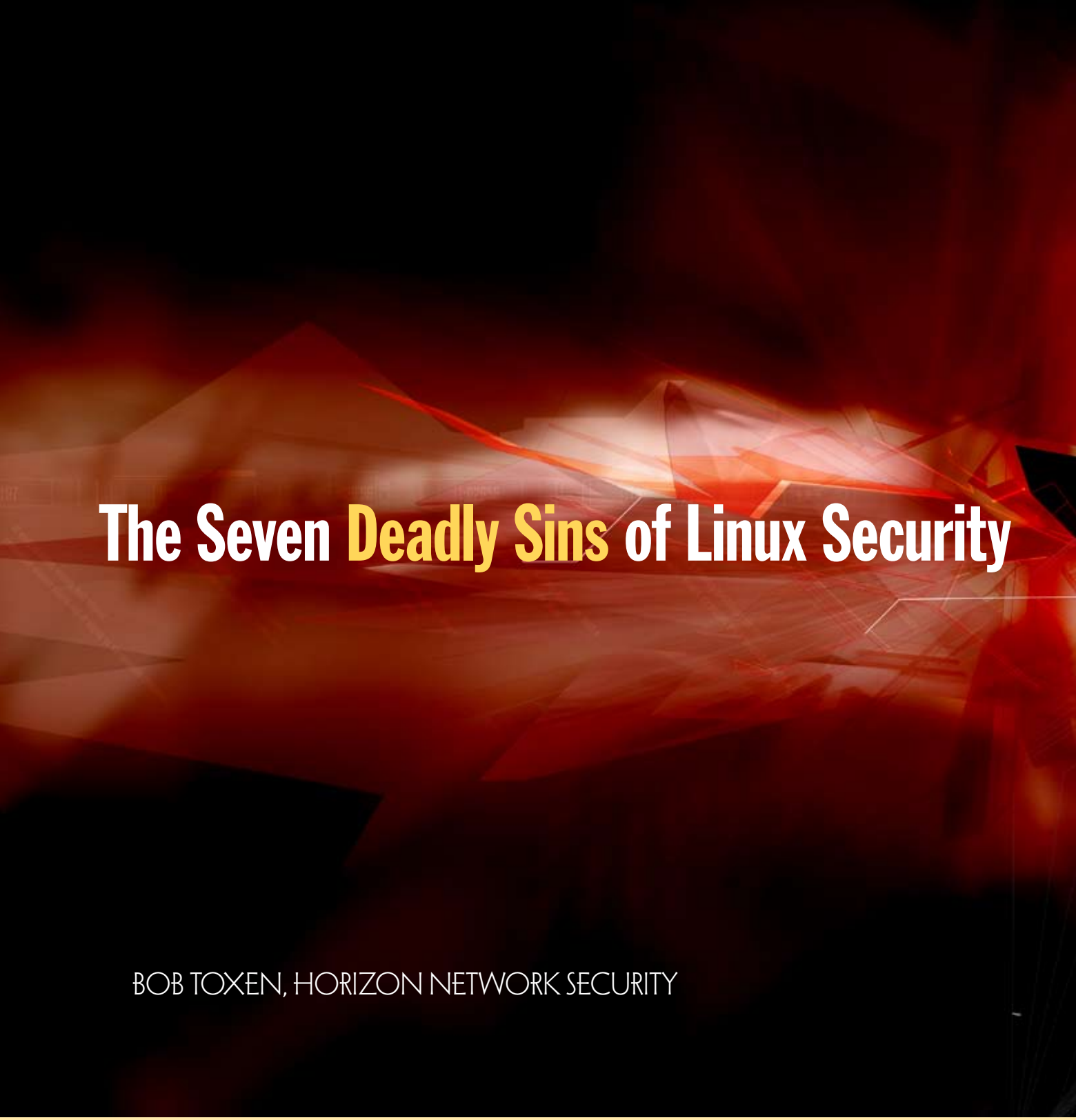

The Seven Deadly Sins of Linux Security

BOB TOXEN, HORIZON NETWORK SECURITY

The problem with security advice is that there is too much of it and that those responsible for security certainly have too little time to implement all of it. The challenge is to determine what the biggest risks are and to worry about those first and about others as time permits. Presented here are the seven common problems—the seven deadly sins of security—most likely to allow major damage to occur to your system or bank account. If any of these are a problem on any of your systems, you will want to take care of them immediately.

These seven deadly sins are based on my research and experience, which includes too many people who wait until after their Linux or Unix systems have suffered security breaches before they take action to increase system security, and on forensics analysis and discussions with systems administrators. Most of these sins and their solu-

tions also apply to Macs, Windows, and other platforms.

They are not ordered by risk level because committing any one of them will likely allow your system to be compromised if it is accessible from the Internet. Even if you are behind a firewall, if you receive any untrusted data from the Internet, such as Web pages, e-mail, or instant messages, your system is at great risk. Avoid these sins like the devil.

Without further ado, here are the seven deadly sins and what to do about them.

SIN ONE: WEAK PASSWORDS

As a systems administrator, you are aware of the system breaches possible on your Linux or Unix machine. You have taken the time and effort to devise a difficult-to-guess root password that uses at least 12 characters that

The Seven Deadly Sins of Linux Security

include at least two words or no words from the dictionary, uses both letters and digits, and has upper- and lowercase letters and some punctuation characters.

I still run into clients with passwords so simple that any hacker could break them in a few minutes with a tweaked version of ssh that guesses different passwords. Such hacker tools can be found on the Web easily with Google or built by any C or C++ programmer. On Internet-accessible systems, I have seen root passwords consisting of a word followed by a small number, where that word is related to the company, what it does, who is in it, or where it is. A good hacker will go to your Web site and see all of this information, then feed it into a password-cracking program.

Another common mistake is to use the same password or very similar passwords for root accounts (or other important accounts) on different systems. Thus, a cracker who breaches one system through a means other than password guessing will then be able to install a Trojaned server for ssh, FTP, or IMAP, or a Trojaned CGI program on that system, see what passwords you use, and try them on the other systems. I have seen this happen many times.

A variation is to use ssh public keys to allow an account on one system to ssh into another system without supplying any password. At the very least, pick a moderately hard-to-crack password for your ssh keys. If you must have an automatic program use ssh without a password to ssh into another system, then create either a separate nonroot account on the target system or an alternate account with UID 0 but a login "shell" that does just what is needed, such as doing a backup.

An even better solution, say for a remote backup, would be for the system needing to be backed up to ssh into the system receiving the backups as a unique unprivileged account for this purpose and copy an encrypted version of the backup. Thus, if the backup server is compromised, no confidential data will be obtained.

Let's hope your root password is awesome and that no one could guess it in 100 years. OK, some obsessive with a program such as Crack could destroy it in a few days except that you use shadow passwords, but that's another story. It is critically important to select good passwords.

How are your users doing? Choke, cough, gag, hack. Every account is a possible entry point. Have your users followed your advice, company policy, or threats to devise good passwords? Are they being as careful as you are? Probably not. Now it is your turn to don the black hat and think like your enemy.

Can you break into your users' accounts by using a password-cracking program? You definitely will need to get written management approval to conduct this level of security audit. There are notable cases of unauthorized audits landing people in jail or at least on the unemployment rolls. (Randal Schwartz is one. The software consultant and author was brought to trial for accessing a password file at Intel in what he says was an attempt to show lapses in security.)

You might even install a module in the passwd program that automatically tries to break a user's proposed new password. Though the standard passwd program

TIP

Consider how severe the consequences would be if one account or one system gets hacked. Can the hacker then get into other accounts or other systems? If so, change passwords, ssh usage, etc. so that the hacker cannot spread the damage to other accounts and systems.

This illustrates the concept of containment. Accept that some account, possibly root, on some system will get compromised. Ensure that the compromise will not spread by doing careful failure analysis now, before you suffer a compromise.

TIP

Protecting every account is critical because of local root vulnerabilities in various programs and the Linux kernel itself. These vulnerabilities allow a hacker who gets shell access as any user to make himself or herself root.

```
passwd password requisite /usr/lib/security/pam_cracklib.so retry=3
passwd password required /usr/lib/security/pam_pwdb.so use_authtok
```

FIG 1

TIP

Avoid default passwords as if your job depended on it.

makes very simple tests, there are more sophisticated routines that include much of Crack's capability. One way to do this is to make use of the cracklib capability in the PAM (pluggable authentication modules) enhancements to the passwd program. The cracklib library analyzes passwords to determine if they can be easily cracked. PAM offers additional security for Linux and Unix systems.

Edit the /etc/pam.d/passwd file to include the code in figure 1. This will cause the PAM-enabled passwd program to load these dynamically loadable program libraries. PAM now is standard with Red Hat. On some systems these are in /lib instead of /usr/lib. (Another good source for PAM information is http://www.sun.com/software/solaris/pam/.)

On Slackware this capability will be enabled if the following line is present in /etc/login.defs (and the dictionary is installed):

```
CRACKLIB_DICTPATH    /var/cache/cracklib/cracklib_dict
```

Consider restricting which remote systems can ssh into your systems' various accounts either through IP tables firewall rules or by editing your ssh server's configuration file, /etc/ssh/sshd_config, to limit which remote systems can ssh in and which accounts they can ssh into, or use both methods for additional security. Make this list very short for root (in sshd_config).

SIN TWO: OPEN NETWORK PORTS

Just as every account on your system is a potential path for a password cracker, every network service is a road to it. Disable and uninstall services you do not need. Most Linux distributions and Unix vendors install tons of software and services by default. They deliberately prefer easy over secure. Many of these are neither necessary nor wanted. Take the time to remove software and services you do not need. Better yet, do not install them to begin with.

To find out which services are being run, use the netstat -atuv command. Even a home system can have dozens of different ports open. A large Web server could have more.

If there are services listed that you do not want to be provided by this box, disable them. Many distributions offer a control panel to do this easily, including Red Hat and Mandriva. You may want to remove the binaries from the disk or chmod them to 0, especially any that are set-UID or set-GID.

NFS, finger, the shell, exec, login r* services (rsh, rexec, and rlogin), FTP, telnet, sendmail, DNS, and linuxconf are some of the more popular services that get installed by default on many Linux distributions; at least some of these should not be enabled for most systems. Most are controlled by the daemon xinetd; these can be disabled by editing the /etc/xinetd.d/* scripts.

You do not need the FTP or telnet daemons to use the respective clients to connect to other systems. You do not need the sendmail daemon listening on port 25 to send mail out, to send mail to local users, or to download mail via POP or IMAP. (You do need to invoke sendmail periodically to de-spool delayed outgoing mail.) You need DNS (named, the name daemon) only if other systems will be querying yours for this data. Most programs running on your own system will be very happy to read /etc/resolv.conf and query the main DNS server of your ISP or organization instead of contacting a named process running on your system. Coincidentally, named's ports are some of the most popular ports that crackers use to break into systems. If you do need to run named, use the recently added facilities that allow it to chroot itself and switch to a nonroot user.

All of these services, except the normal installations of NFS,[1] DNS, and sendmail, are started on demand by xinetd. They may be turned off by commenting out their entries under /etc/xinetd.d. Many distributions offer a control panel or Linuxconf to do this easily, including Red Hat and Mandriva.

The Seven Deadly Sins of Linux Security

The stand-alone services are turned off by altering their entries under /etc/rc.d or in configuration files there.

On Red Hat-based systems, issue the following commands to shut down portmap and prevent it from being restarted on reboot.

```
/etc/rc.d/init.d/portmap stop
chkconfig --del portmap
```

An alternative tool is the ASCII menu-based ntsysv program. Like chkconfig, ntsysv manipulates the symbolic links only under /etc/rc.d/rc[0-6].d, so you also will need to explicitly shut down the service. To do both of these, issue the commands

```
/etc/rc.d/init.d/portmap stop
ntsysv
```

On other distributions that use System V-style startup scripts (/etc/rc.d/rc[0-6].d directories for Red Hat derivations and /etc/rc.[0-b].d for Debian), rename the appropriate script under rcX.d (X usually is 3) that starts with a capital *S* and has the service name in it. For example,

```
cd /etc/rc.d/rc3.d
mv S11portmap K11portmap
```

Just as only scripts starting with *S* are invoked when entering the respective run level, scripts starting with *K* are invoked when exiting that run level. This is to turn off daemons that should run only in that run level. For example, this mechanism will turn off sshd, the ssh daemon, when switching from run level 3 (multiuser with networking) to run level s (single-user mode). Just as a selected S*something* script can be disabled by renaming to s*something*, one of these latter scripts can be renamed from K*something* to k*something* to disable it.

On Slackware and similar systems, simply comment out the lines starting them in /etc/rc.d/*. The grep program may be used to find these. Be sure to terminate any of these services that are running on your system after altering the configuration files.

If you do not want to bother with kill, a simple reboot will do this and verify that the configuration files were correctly altered. (Having a set of available rescue disks before this reboot would be a fine idea.)

To remove these services from your system, you can use your distribution's package manager. Red Hat-based installations use RPM; Debian-based distributions use dpkg; SuSE uses YAST; and Slackware uses pkgtool.

Linux and Unix are like the Swiss army knife of networking: they have one or two tools that get used all the time, others that are used less often, and some that are never used. Unlike the Swiss army knife, you can slim down Linux or Unix to just the services you need and discard those you do not. I will never use the awl or scissors on my knife just as I will never use rsh or the set-UID to root features of mount or umount.

Decide which ports you wish to have open (such as www and ftp) and close the rest. Closing unnecessary ports makes your system more secure and perform better.

> The most careful sysadmins will reboot their systems several times after making changes to startup scripts, other configuration files, and the kernel, and after installing security patches to ensure correct and reliable startup and operation.
>
> TIP

SIN THREE: OLD SOFTWARE VERSIONS

Linux and Unix are not perfect. People find new vulnerabilities every month.[2] Do not despair, though. The speed with which problems are found and fixed in Linux is the fastest on the planet. Your challenge as an administrator is to keep up with the changes.

Each distribution has a mailing list through which security bulletins are issued, and an FTP or Web site where the fix will be available. There are also excellent independent security mailing lists, such as Bugtraq and X-Force's Alert. You can (and should) subscribe to these lists.[3]

Other good sources of Linux security information are http://www.lwn.net/ and http://www.linuxtoday.com/. These sites are distribution-neutral and carry all of the major distributions' security advisories.

One of the advantages of Linux is that when a fix is issued, it is very quick to install. Furthermore, unless it is in the kernel, your downtime for that service is on the order of seconds or minutes. Rarely, if ever, is a reboot necessary.

SIN FOUR: INSECURE AND BADLY CONFIGURED PROGRAMS

The use of insecure programs (such as PHP, FTP, rsh, NFS, and portmap) in other than carefully controlled situations and failure to configure other programs properly continues to be a major security sin.

Most sysadmins know that POP and IMAP (unless wrapped in SSL), telnet, and FTP[4] send passwords and data in the clear (unencrypted). They know that PHP, NFS, and portmap have a history of security problems, as well as design defects in their authentication. Many use them anyway, and then are surprised when they get broken into. Instead, use spop, simap, ssh, and ssh's scp or sftp, or put a good firewall in front of that subnet, or set up a restricted VPN between your facilities. If you absolutely must use PHP, keep it patched and carefully audit your code for problems.

Many programs are secure only if properly configured. It is common for sysadmins to configure them improperly, sometimes because of a lack of training and understanding of the risks; other times use of an insecure feature is deliberate, because "I just gotta have it." A recent case in point is Apache's PHP capability, which has had a history of security problems. These problems have been well publicized, and still some people cannot seem to use it securely or find an alternative. Security and convenience are often contradictory, and you have to make a choice between the two.

Before deciding to deploy a service (or changing which capabilities will be used or how the service will be deployed), do some research. Check the security history and understand how the service may be deployed securely. If it cannot be deployed securely, what are secure alternatives? I still encounter people using FTP, not realizing that sftp is an excellent alternative. Putting an insecure service such as NFS behind a firewall may be the solution for some. For others, putting their insecure Windows networks behind firewalls, with their different offices linked via a VPN between these same Linux firewalls, offers excellent security. Configure a firewall with separate subnets on separate interfaces for different categories of users, such as students and faculty or sales, human resources, and engineering.

Absolutely prohibit wireless networks inside of the firewall or to any system with confidential information unless all wireless traffic first is encrypted with IPsec or equivalent. Do not rely on WEP (Wired Equivalent Privacy) or its successors.

Web servers and CGI programs are the bane of Linux and Unix computer security. Simply speaking, a CGI program is one of the easiest ways that a hacker can get into your system. It is essentially a program that runs on your computer at the request of anyone and everyone without passwords and has the access to do powerful things (for example, shipping valuable merchandise, revealing confidential data such as your customers' credit card numbers, and moving money between accounts).

A CGI allows anyone to access your Web site, good intentions or not. While other "accepted" servers such as sendmail and named also will talk with anyone, the scope of what a client may request is far smaller. Although these latter servers have had their share of serious security bugs, those that keep their security patches up to date have minimal risk.

Here are a few hard and fast rules that will help make your Web site secure.

Know your data (supplied by Web clients).

- Establish maximums and minimums for data-entry values and lengths of fields.
- Decide which characters are acceptable in each field. Expect the malicious to send you control characters and non-ASCII bytes. Expect that crackers will use the % encoding or alternate character sets to generate these evil characters. Thus, you need to check for illegal characters both before and after % conversion and in different character sets.
- Double-check each entered value. A surprising number of shopping-cart packages put the price of items in the form and believe the price in the filled-out form sent by the user. All a user needs to do to give himself or herself a discount is to alter this form.
- If possible enumerate the allowed values instead of using ranges (except for listing ranges of letters and digits).
- Understand, too, that an evil Web client can send bytes back to your server. The hacker may copy and alter your Web form to change your "fixed" fields, etc.
- Use a secure language. Client-supplied data never should be handed directly to a shell script; there are too many opportunities for a cracker to get a shell or to exploit a buffer overflow vulnerability. For many that secure language will be C, C++, Perl, Java, or Python. If that language offers checking for tainted data, use

it. One language does not fit all. Perl has a number of features to enable safer CGI programs.[5] These include the "tainted data" feature, the -w flag to warn you about things that you are creating but not using, the strict capability, and perlsec. These features are discussed in http://perldoc.perl.org/perlsec.html.

- If you have many CGI programs—with a few being carefully written so that they manipulate confidential data, and some that are more casually written because they do not handle critical data—consider the following. Use the suEXEC program that comes with Apache to run these different classes of CGIs as different Linux or Unix users. This allows you to use operating system file permissions to prevent the less-trusted CGIs from accessing more confidential data. Documentation on suEXEC is available at http://apache.org/docs/suexec.html.

Analyze and audit CGIs for vulnerabilities.
When writing CGI programs, look at them the way a cracker would and try to break them. Stop buffer overflows by using good programming techniques. An easy way to determine if the line is larger than the buffer is to see that it does not end with a newline character, as this example illustrates:

```
#include <stdio.h>
#include <string.h>

int   c;
char  buf[200];

if (!fgets(buf, sizeof buf, stdin))
        error();
else if (!strchr(buf, '\n')) {
                /* Read rest of long line. */
        while ((c = getchar()) != EOF
          && c != '\n')
                ;
        overflow();
}
```

Do not use the gets() routine because it does not do any checking for buffer overflows; use fgets() instead. Many of the other popular C string functions have similar weaknesses. The strcpy() function, for example, "lets" you copy a large buffer into a small buffer, overwriting unrelated memory. The strncpy() function is an excellent alternative. A safe way to copy strings is:

```
strncpy(dest_buf, source_buf,
  sizeof dest_buf);
dest_buf[sizeof dest_buf - 1] = '\0';
```

To detect a problem, one possibility is:

```
if (strlen(source_buf)
  >= sizeof dest_buf)
  error();
else
  strcpy(dest_buf, source_buf);
```

Check for escape sequences, the possibility of a client issuing Linux or Unix commands (by inserting spaces, quotes, or semicolons), binary data, calls to other programs, etc. Often it is safer to have a list of allowed characters rather than determining each unsafe character.

The following C code may be used to process a field in which the client should supply his or her name. In this example, the calling process supplies a NUL-terminated string; this routine returns 0 if the string is a legal name, and -1 otherwise. The second argument specifies the maximum legal string allowed, including the terminating NUL byte. Note that the calling routine must be careful to ensure that its buffer did not overflow. I chose clear code over slightly more efficient code.

```
#include <string.h>

char  okname[] = " .'-,abcdefghijklmnopqrstuvwxyz"
                 "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

/* Return 0 on legal names, -1 otherwise. */
legal(char *name, int maxlen)
{
        if (!name || !*name
          || strlen(name) >= maxlen)
                return -1;
        while (*name)
                if (!strchr(okname, *name++))
                        return -1;
        return 0;
}
```

Many system break-ins relating to Linux and Unix Web servers happen via insecure CGIs.

Implement rings of security in CGIs.
Try to design your application so that even if it finds a CGI vulnerability, the system is protected from major damage. One solution is to have CGIs just be front ends for a solidly written server running on a different machine. The more hurdles a hacker must jump to reach the goal, the more likely it is that he or she will stumble.

Watch for bug reports in third-party CGIs and inspect their code.
If you use third party-supplied CGI scripts (such as shopping carts), you should subscribe to the appropriate mailing lists and watch for security bulletins. If possible, get the source code and review it. If you do not know the language, then get someone who does to review it.

Many CGIs, both commercial and open source, have severe security holes that are well known to the hacker community. Many locally written CGIs have security vulnerabilities because the programmers who write them typically have no training in writing secure code and such code is rarely audited.

Avoid creating and using set-UID and set-GID programs to the maximum extent possible, especially programs set-UID to root (and try real hard).
Many system programs run as root. Frequently all these programs need to be set-UID to run as some user to gain access to data that should not be world accessible. Other programs need to be set-UID to root only when starting to open a low network port for listening or to change its privileges to that of a particular user. In this case, the program then should give up root privileges. Apache, named, and ftpd were enhanced several years ago to do this for better security. Different programs may need to be set-UID to different users to protect them from each other.

Do not keep clients' confidential data on the Web server.
Avoid storing users' privileged data (credit card numbers, financial details, mailing addresses and phone numbers, etc.) on the same machine as the Web server. This separation will force a hacker to crack two systems instead of just one to get this data.

Do not include users' confidential data (credit card numbers, financial details, mailing addresses and phone numbers, session ID, etc.) in an URL or cookie.[6]
Frequently this is done (insecurely) as arguments to a CGI program. Consider the following example:

```
www.abroker.com/cgi-bin/address_change?account=666
?passwd=secret&addr=1+Maple+St.&phone=301-688-6524
```

Some browsers may store this URL (containing confidential data) in a history file. If someone is browsing from a public terminal, such as a school or library, you could be liable for careless handling of the data. Similar issues are present for cookies.

Be very sure that the privileged data that a user supplies on a form does not show up as the default data for the next person to "pull down" that form and see.
Yes, this has actually happened.

Always protect the user who types in a password.
Take the user to a secured area prior to this information being entered and ensure that the password or credit card number will be encrypted on the system (with https) before transmission to your server.

SIN FIVE: INSUFFICIENT RESOURCES AND MISPLACED PRIORITIES

At many organizations, management simply will not approve sufficient resources to allow sysadmins to provide good security. It takes many things to achieve a truly comprehensive security solution. Education, design, proper implementation, user training, maintenance, and continual vigilance all are required for an organization to be secure. Frequently, security is limited to what a sysadmin is willing to do on his or her own time. Yet, a sysadmin who is unwilling to spend the time will certainly be blamed for any violations. This deadly sin concerns problems that are not the sysadmin's direct responsibility. In other words, management will not allow the sysadmin to make the changes necessary for good security.

This may not be a "technical" problem, but it has been the cause of break-ins at numerous organizations. Lack of resources commonly is a result of misplaced priorities. For example, the following is a common misconception of those whose organizations have not been broken into: "The media exaggerates every danger well beyond the true risk." Show your manager media accounts of large companies that have suffered security breaches. If you shopped at T.J. Maxx or Marshalls in 2006, you probably received a new credit card number thanks to TJX Cos., the parent company, which suffered a security breach in December. Circuit City suffered a similar breach. Consider making a present of Bruce Schneier's excellent

The Seven Deadly Sins of Linux Security

book, *Secrets and Lies: Digital Security in a Networked World* (Wiley, 2004), to your boss. *Secrets and Lies* is aimed at management and limits the tech-speak.

On a number of occasions, I have warned clients about major security problems only to have them decide that security was not as important as getting that next release out or making nonsecurity-related computer improvements. Later, they learned the sad reality—recovering from a security breach commonly costs 10 times as much as having implemented good security before the break-in—and only then did they spend the money to implement the security.

Furthermore, the estimate of the cost of recovering from a security breach being 10 times the cost of prevention is only the direct cost. It does not account for the lost market opportunities for delayed products, the loss of customers who heard about the security breach and went elsewhere, and the costs to customers and employees who could not access your Web site and e-mail during recovery. It does not account for lost investors and other consequences of bad publicity, and it most certainly does not account for the damage done to an IT professional's career.

What can be done to resolve insufficient resources and misplaced priorities? Spend an hour or two a week working on security as a skunk-works project.[7] Demonstrate a Linux firewall, Web server, or VPN. Show how easy it is to update Linux software when patches come in, to use ssh and gpg, to crack most passwords, or attack a Wi-Fi wireless network. Do scans of your network from your home system (using nmap with the -O flag) to show how open your network is. Install Snort and PortSentry outside of your firewall (if any) to show how often your network is attacked.

Make a point of talking with your colleagues to get detailed accounts of problems that you can then relay to your management. Have a good consultant or other trusted outside source do a security audit of your company and recommend improvements. Giving up leads to procrastination, and procrastination results in compromised systems. That is the dark side of The Force. Never give up. Never surrender.[8]

Misplaced priorities can also mean using Microsoft because "We are a Microsoft shop," disregarding that it may not have sufficient security for servers accessible from the Internet.

SIN SIX: STALE AND UNNECESSARY ACCOUNTS

As discussed before, each account is a possible entry point into the system. A stale account's password will not be changed, thereby leaving a hole. If the account has data that needs to be reassigned, disable the account by putting a * or !! in the ex-user's password field (after the first colon) in the /etc/passwd file. This disables logging in via that account because no password encrypts into either of these values and shadow password-enabled code understands these sequences. Get things cleaned up as soon as possible. Make sure that no set-UID or set-GID programs or publicly readable or writable files containing confidential data remain in that account.

Issuing the following code

```
chmod 0 /home/someone
find / -user someone -ls
```

is a good start. Note that the user may have a mailbox, files in the print spool directory, accounts in various applications, etc. that will need to be attended to.

Some of the services you removed (while correcting an earlier sin) have accounts in the /etc/passwd file. When you remove that service, make sure that the /etc/passwd account also is removed or disabled. Some of the notables are FTP, NFS, uucp, mail, gopher, and news. If you do not need them, get rid of them.

SIN SEVEN: PROCRASTINATION

In many reports of intrusions the sysadmins say, "I meant to install... IP Tables... TCP Wrappers... a newer version of... a firewall... turn off NFS and portmap... stop using PHP..." Clearly they knew, at least vaguely, what had to be done but delayed until it was too late.

Sure, you have more responsibilities than time, but consider setting aside an hour twice a week to upgrade security. Those hours may come with bag lunches at

When a user will no longer be using the system, be sure to remove his or her account from the system quickly.

your desk, but that beats a cot in your office so that you can work around the clock for a week recovering from a compromise. Sadly, I know of one company where they did bring in those cots for a number of engineers during a weeks-long recovery project following a breach. Worse, they procrastinated on deciding to build a firewall until after this event. Q

ACKNOWLEDGMENTS

This article is based on *RealWorld Linux Security: Intrusion, Detection, Prevention, and Recovery*, second edition, by Bob Toxen (Prentice Hall PTR, 2003, ISBN 0130464562); chapter 2, section 2, "The Seven Most Deadly Sins."

Thanks to Prentice Hall PTR for granting permission to use material from the book in this article. Thanks to Larry Gee, a very talented programmer, for co-authoring this section of the book.

REFERENCES

1. NFS consists of these daemons and a few more, including: rpc.nfsd, rpc.mountd, portmap, rpc.lockd, rpc.statd, rpc.rquotad, and automounter, scattered among a number of startup scripts. A cracker process can lie to portmap and masquerade as a legitimate server. NFS has had plenty of security bugs in the past, and its design prevents it from being made secure in many configurations.
2. Most recent vulnerabilities are not directly exploitable remotely on most systems. This means that most systems are not at risk for remote attack from the Internet. Many of the vulnerabilities may be taken advantage of by someone with a regular shell account on the system. Others are in programs that most people do not use and that are not set-UID or set-GID and thus are not a threat.

   This is different from most Windows vulnerabilities where almost every client system or server using that major version of Windows is vulnerable to remote attack over the Internet and thus to complete control by crackers. We observe that most Windows vulnerabilities affect all Windows versions released in the past four years, including Vista. We have recently seen Vista included with past versions of Windows for several remote "root" vulnerabilities.
3. Subscribe to Bugtraq by sending e-mail to bugtraq-digest-subscribe@securityfocus.com with empty subject and content. Subscribe to X-Force's Alert by logging on to https://atla-mm1.iss.net/mailman/listinfo/alert.
4. If you are doing only anonymous FTP, your password is normally your e-mail address. Unless you are a government researcher at Groom Lake (Area 51) and you do not want to acknowledge the existence of such a facility, then generally you have nothing to worry about.
5. Most of the information on Perl presented here is from Kurt Seifried's writings.
6. Fidelity Investments, which manages $900 billion of its customers' money, did not follow this advice. In May 2002, it was reported that by changing the digits in the URL of the page displaying his statement—a three-digit number—a client saw other clients' statements.
7. A skunk-works project is one done in secret without management approval or knowledge.
8. Thanks, *Galaxy Quest*.

TIP

The Linux 2.6 kernel prior to 2.6.17.4 has a nasty local root vulnerability where anyone with a shell account, possibly via ssh or abusing a Web server CGI program, can make himself or herself root. See CVE-2006-2451.

Are any of your systems vulnerable to this right now? I thought so.

A partial fix is to issue the command:

```
chmod 700 /etc/cron*/.
```

A better solution is to write a kernel-loadable module to prevent use of the prctl() system call by other than root.

Of course, the only full solution is to upgrade your kernel. If the system is at a remote office or colocation facility where there are no experienced sysadmins, then good luck if the new kernel does not boot.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

BOB TOXEN is a computer and network security consultant with 33 years of Unix experience and 12 years of Linux experience. He was one of the 162 developers of Berkeley Unix and one of the four creators of Silicon Graphics' Unix. He has been an advisor to the George. W. Bush administration on computer issues at the five principal intelligence agencies.