UnixWorld Online: Internet Security: Column No. 003

Auditing your Unix System, Part 1

By Rik Farrow

Auditing Unix systems is not an easy task. This article examines some of the issues, free and commercially available tools, and looks at the password file.

Questions regarding this article should be directed to the author at *rik@spirit.com*.

The Unix operating system has a terrible reputation when it comes to security. Just how bad became clear to me when I sat in on a hackers' conference (DEFCON I) in Las Vegas. One of the organizers boldly stated, ''Unix is our friend''. What he referred to wasn't that Unix systems were poorly designed, but rather that Unix systems possess a rich environment for crackers. You can tighten the security of your Unix system by auditing the system and correcting configuration mistakes. You should also keep abreast of security patches announced by your Unix systems' vendors.

The Unix system was designed by programmers for programmers. Initially, one of the most compact and elegant multi-user and multitasking systems, Unix systems are fattened up with the additions of many features regarded as essential today, including most significantly--networking. So what was once a concise development environment--for bright programmers--which would run in 128 kilobytes of RAM has become a complex operating system for mulitprocessor systems running as servers. When I installed Solaris 1.2 (also known as SunOS 4.1.4) on my SPARCstation, I counted (using `find / /usr -xdev -print | wc -l`) almost 14,000 files! And that would have been larger had I included several other optional subsystems. The immense number of files, programs, and directories makes auditing Unix systems a daunting task.

There are tools for making this task simpler. In the Unix tradition, free software known as COPS (Computer Oracle and Password System), can be acquired and locally installed. COPS does a good job of automatically auditing a Unix system for security. COPS doesn't fix anything, but does create a terse report of what it considers to be problems. Building COPS may be difficult if you are not familiar with building freely available Unix software. You can collect the source code to COPS from CERT (Computer Emergency Response Team) by anonymous FTP. Of course, this site has many other security-related tools.

Other good places to visit include:

Purdue University's COAST security archive site (ftp://coast.cs.purdue.edu/pub/COAST)
: This site has SATAN, Tripwire, scan-detector, and many other tools. There are also policy statements used at Purdue and a paper about how Internet policy was created at Intel (`horses.ps.Z`).

Texas A and M University's FTP site (ftp://net.tamu.edu/pub/security/TAMU)
: This site includes Tiger, an auditing tool that uses digital signatures to check that your

workstations contain only patched, up-to-date versions of software, a white paper about the attack that lead to the creation of Tiger, and other tools. You may wish to check out `tiger.README`, `tiger-2.2.3.tar.gz`, and `tiger-2.2.3.check_dev`.

You can also buy audit software from companies such as Open Vision (which bought Demax), Axent, DEC, or VESoft. Commercial auditing packages have two advantages over freeware. For one, a commercial package is pre-built and easier to install. Also, most commercial packages are designed to audit groups of Unix systems that are connected to networks, making the auditor's task easier. Some audit tools will also create command files (shell scripts) of suggested changes, making it simpler to correct possible deficiencies. We've supplied vendor contact information.

Not that commercial systems are perfect either. The only package I've had an opportunity to review has been SecureMax, which I liked. I've seen some output from Axent's newest product while it was still in Beta, and--in my opinion--it needed work. Your mileage may vary.

Some recommendations made by auditing packages may actually ''break'' the system. For example, COPS suggests changing permissions for Solaris 1.x device files that represent file systems in such a way that the disk free command (`df`) won't work for ordinary users. Open Vision's SecureMax will change permissions on the Sun system `/etc/utmp` file so that error messages are displayed when the window system starts up. However, in this last example, restricting access to `/etc/utmp` is the correct security action to take.

COPS also reports that the permissions on `/etc/utmp` are incorrect. (This file contains the identity and port address of currently logged-in users.) One cracker attack modifies this file to hide the fact that an account is in use, while another uses this file to overwrite other files when commands like `wall` are run.

You should also keep up with your Unix vendor's security announcements. If crackers are using a flaw in delivered software, vendors will eventually produce a patch to correct the security hole. Some vendors respond quicker than others, but you can be sure that by the time the vendor has announced a patch, the cracker community has known about the problem for some time. Contact your vendor, learn how to be advised of new patches, and install security patches as soon as they become available.

Rolling Your Own

You can perform your own Unix system audits, something I recommend that you do. It is difficult--if not impossible--to really appreciate the functions and advice provided by audit tools without having gone through the process yourself. While doing so won't make you a security expert, you will learn more about Unix systems and what to expect from a properly configured one. You will need some basic Unix system expertise: how to execute commands, list directories, and understand file permissions at a minimum. As this article is only the first in a series on auditing Unix systems, we'll devote the rest of this installment to the all important password file.

Like any multi-user system, Unix systems have user accounts. These account entries are stored in the password file (`/etc/passwd`), which is a good place to begin an audit. Account entries have seven fields (see Listing 1A) and begin with the account's (log-in) name, followed by a colon (the field separator) and an encrypted password. Encrypted passwords will be 13 characters in length. Sometimes, a single character, such as an asterisk, will replace the encrypted password. If the password field is blank (two adjacent colons) the corresponding account has no password and one should be added. No user account should be without a password, *especially guest accounts and special system accounts*. Special system accounts include `sync`, which normally invokes `/bin/sync`, but it can be abused, and even more importantly system accounts such as adm, bin, daemon, uucp, and so forth. Leaving these accounts without a password just begs for trouble.

Duplicate user-id numbers represent another problem. Each ordinary user account should have a unique user-id number, typically in the range of 100 to 30,000. If two accounts share the same user-id, they also own the files belonging to the other account because file ownership is based on the user-id number, not the account name. Also, duplicate user-ids compromises accountability on your system because users' activities--if monitored with auditing--are labeled with the numeric user-id. I have seen systems where everyone logged into an account bearing a unique user name, but everyone shared the same user-id. Even the home directory was the same. The system administrator might just as well have had a single user account as far as accountability goes.

You can sort the password file using a command line like `sort -t: +2n -3 /etc/passwd` to produce a listing ordered by the user-id field. You can also use the `pwck` (spelled differently on SCO and AIX systems) utility, which performs several password-file checks, including: field counts, user-name length check (should be no more than eight characters because that's the maximum length assumed by most software), the existence of the home directory as well as the program--usually a general purpose shell command interpreter--that is invoked after the user logs in. Listing 1B shows some sample output.

Any program executing with user-id zero has unlimited access to any file or program on the system. The root account is generally the only legitimate account with this access privilege. A common system cracker tactic is to install a bogus account-- with a name different from ''root''--that has this privilege.

A ''hidden'' or ''shadow'' password file is often used in conjunction with the conventional password file. When a such a file is employed, no encrypted passwords will be present in `/etc/passwd`. Instead, the password will be stored in the alternate password file, whose permissions are adjusted so only the superuser can read it. SVR4 uses `/etc/shadow` and SunOS employs `/etc/security/passwd.adjunct`. Listing 1C shows what a password file looks like when used in conjunction with the `passwd.adjunct` file.

Why hide passwords if they're encrypted? Even though such passwords can't be unencrypted by any known algorithm, they can be guessed. Password cracking software uses one or more dictionaries or other lists to provide words used for guesses. Such programs often use an efficient encryption algorithm to encrypt guesses quickly and then match them to the encrypted passwords from a purloined password file.

The `/etc/passwd` file must be readable by all users so programs like `ls` and `find` can use this file to convert user-id numbers into account names. This read requirement was the motivation for hidden password files. By moving the encrypted passwords to an unreadable ''hidden'' file, it is more difficult for crackers to collect encrypted passwords for dictionary-based password-cracking attempts.

Directories Critical

Even though the password file has seven fields, we'll only discuss the home directory (sixth) field further. The home directory is the initial current directory for whatever process is spawned when the user logs into the account. This directory generally contains configuration files, generally known as initialization or startup files, for establishing the initial behavior for certain programs. Some programs that use this approach include the different shells, mail and news reading software, visual editors such as `vi`, and certain file-content display programs, such as `more`.

These initialization files are often hidden, that is, they don't appear in a directory listing made with `ls` unless a special option (`-a`) or form of the file argument (say, `.[a-z]*`) is specified. These files are hidden because their name begins with a dot or period, which isn't matched by default. The term ''dot file'' is used sometimes because the name begins with a dot.

A user home directory and any startup files in that directory should be owned and writable by the account user. The ability to write in the user's home directory or startup files gives a cracker means to create a Trojan horse for that user. For instance, say the cracker installs a program in the home directory, which when executed performs an unexpected function that compromises security. Many attacks on user accounts rely on modifying these startup files, and protecting the home directory is just as important. Setting a safe shell command-search path-- defined by the `PATH` environment variable--that only lists system program directories and not the current directory or ''dot'') also protects against Trojan horse attacks.

While you are visiting home directories, look for two other problematical startup files, `.netrc` and `.rhosts`. The `.netrc` file is used by FTP clients to automate log in to remote FTP servers. Unfortunately, to do so this file must contain unencrypted passwords and if read, gives the cracker access to all the remote systems listed in this file. The `.rhosts` file in home directories pose a related threat. Users employ this file to grant passwordless access to their account to other users over the network. The original intent was to permit users with accounts on several systems access to each system without re-entering a password. But this mechanism lets your users control who has access to your system and so is too dangerous to use.

Going Deeper

This article is the first part of a series about auditing Unix systems. But permissions and networking have hardly been touched upon, and there are other issues not covered. You can learn more about these issues by reading one of several good books about Unix security. My book, *Unix System Security* (Addison Wesley, 1991), is a good beginner's book. David Curry's *Unix System Security* (Addison Wesley, 1992) covers similar material for somewhat more advanced readers. Garfinkle and Spafford's *Practical Unix Security* (O'Reilly and Associates, 1991) is recommended for more advanced readers. And Cheswick and Bellovin's recent book, *Firewalls and Internet Security* (Addison Wesley, 1994) should be required reading for anyone planning to connect to the Internet.

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content

Search

Last Modified: Wednesday, 30-Aug-95 07:19:42 PDT