Sharon A. Kapilow* and *Mikhail Cherepov *are members of technical staff in the Computing Planning Department at AT&T Bell Laboratories in Whippany, New Jersey. Both work on development of UNIX® system security software tools and policies. Ms. Kapilow received a B.S. in mathematics and computer science from Wilkes College and an M.S. in computer science from Rutgers University. She joined AT&T in 1980. Mr. Cherepov received a B.S. and M.S. in computer science from Illinois Institute of Technology. He joined AT&T in 1983.*

QUEST— A SECURITY AUDITING TOOL

Quest is a security package for assisting system administrators of UNIX® systems. It consists of a series of components that check for potential problems on these systems, such as insecure write permission settings on files and directories; login identifiers with questionable aging information and null, old, or matching passwords; unauthorized access to superuser status; and potential "Trojan horses." This paper discusses each component and the problems it detects, describes current work on Quest, compares the package to other security-auditing tools, and discusses future work that will enhance Quest.

Introduction

The UNIX operating system has many features that will provide reasonable security on a computer, such as permissions, passwords, `umask` (which allows users to specify default permissions for files and directories they create), and log files.[1] It is up to computer system administrators to apply those features to maintain the integrity of their systems and to devise procedures to audit the use of the features to ensure that they are used properly. Security software packages have been developed to automate the procedures and thus help administrators in their efforts. Quest is one of the packages developed for this purpose and is the subject of this report.

This paper assumes that the reader has some knowledge of the UNIX operating system. It is not meant to be a tutorial on computer security; rather, it discusses UNIX system security issues and ways in which AT&T is addressing them.

Quest was developed by the R&D Computer Security Group at Whippany, New Jersey, and is based on the Quaestor program written by Fred Grampp of the AT&T Bell Laboratories Computing Science Research Center, Murray Hill, New Jersey. Quest was proposed to improve Quaestor, with the goal of making it an AT&T commercial product.

Overview

Quest currently consists of several components that check for the following problems on UNIX systems:

- Login identifiers (IDs) with missing aging information or expired aging information; null and old passwords; and easily guessable or derivable passwords
- Files and directories with insecure write-permission settings (that is, files or directories that are writable by unauthorized persons)
- Unauthorized and unsuccessful use of the `su(1)` (superuser) command
- Potential Trojan horses
- Insecure `umask` settings and permissions

of users' `.profiles` and PATH entries.

These problems, detailed in this section, are caused by improper administration and careless or uninformed users. They are only a subset of the total number of security concerns on UNIX systems but are significant enough to warrant checking by a system administrator on a regular basis.

Note that all security solutions mentioned in this report are concerned with preventing outside as well as inside attacks—that is, preventing outsiders from accessing the system and preventing anyone who has access to a system from stealing information or damaging the system in any way. For the purposes of this report, all such persons will be grouped under the term "intruders."

Insecure Login IDs. One of the biggest concerns in computer security is unauthorized access to login IDs on the computers. Administrators must keep outsiders from accessing their systems and keep current users from obtaining unauthorized privileges. The UNIX operating system provides three features to keep login IDs secure:[2] passwords, one-way encryption, and password aging. However, to take full advantage of these features, certain guidelines must be followed by both users and system administrators. Quest checks the entries in the password file and identifies those that do not meet the standards for a secure login ID. For example, Quest flags the following typical situations as problems:

- Login IDs with null passwords. A login ID having a null password entry may be accessed via `login(1)` and `su(1)` commands without a password being supplied. Thus, *anyone* who knows such a login ID and its relevant system can log in.
- Login IDs with no aging data. Passwords expire only when aging data are present; otherwise, a password may be used indefinitely. A password that is retained for long periods increases the chances for an intruder to crack it and gain access to the system.
- Login IDs with old passwords. An old password is one that has been expired for more than some predefined period of time. It may indicate an abandoned login ID. Such IDs provide a convenient means for an intruder to attack a system without being detected.
- Login IDs having passwords that are derivable from information stored in other fields in the password file. Password guessing programs have been written that use data gathered from the information in a user's entry in the password file.
- Login IDs having passwords that are common male or female first names. The use of such passwords lets intruders breach systems merely through brute force (for example, by trying every common male and female name as a password for each login ID in the password file).

Many attacks on a system are started by an outsider who gains access to the system through an insecure login ID or a user who gains more privileges through another user's login ID. Since login IDs and passwords are the primary defense mechanism in the UNIX system, checking them is an important part of good system administration. With Quest, the checking can be done automatically.

Insecure Files and Directories. Once someone has access to a system (either by virtue of being a user on the system or getting on through an insecure login ID), he or she can attempt to compromise the system by getting more passwords, stealing information, and damaging the system. Protecting write permission on system files and directories is a key step in preventing such actions from occurring, since an insecure file or directory permits arbitrary changes.[3]

Quest checks all system files and directories and identifies any that have insecure write permissions. These are defined as files and directories that are writable by someone

Panel 1. Examples of Insecure Files and Directories

Example 1:

```
    -rwxrwxrwx    1 root      bin       33623 Sep 13 11:34/bin/who
```

Example 2:

```
    drwxrwxrwx    9 bin       bin       4408 Sep 21 15:25 /etc
```

Example 3:

```
    -rwsrwxrwx    1 root      bin       23041 May 21 17:31/bin/mkdir
```

other than `root` (the supervisor), `root`'s group, or a predefined list of users and groups that are equivalent to `root`, in the sense that anyone who can gain access to the user or group ID can eventually become `root`.

Examples of files and directories that Quest will identify as "insecure" appear in Panel 1. In Example 1, the routine `/bin/who` is writable by "other," which means that everyone can overwrite the system's `who` routine with his or her own executable routine (or simply, "executable"), which is likely to be run by innocent users. An intruder can take advantage of this in the following way: The intruder replaces `/bin/who` with his or her own version of `who`. This version creates a file that, when executed, runs as if the owner of the file were executing it. (Such a file is known as a *setuid* file.)[4] When a user executes `who` to determine who is on the system, the bogus `who` routine creates a setuid file owned by the user and executable by the intruder. The intruder can then execute this file and impersonate the user.

Example 2 shows a directory with insecure write permissions. A user who has write permission to a directory can add and delete files from the directory, even though the user may not have write access to files in the directory. In this example, `/etc` is writable by "other," which means that everyone can delete any file in the directory (such as `/etc/passwd`) and replace it with his or her own version of that file. In this case, `/etc/passwd` can be replaced with a new password file containing an entry with no password and a user ID of 0, which is a superuser's ID. The intruder can then log onto the system as a superuser without typing a password.

Example 3 shows a setuid file with insecure write permissions. In this example, `/bin/mkdir` is setuid to `root` (owner of the file) and writable by "other." This means that everyone can overwrite `/bin/mkdir` with his or her own executable and run it (as `root`) to assist in an attempt to compromise the system. An example of such an attack would have an intruder copying `/bin/sh` into `/bin/mkdir` and executing `mkdir` to become `root`.

As can be seen from the examples, write permissions on system files and directories are critical and must be checked regularly. With Quest, this checking can be done periodically and easily.

Unauthorized su(1) Usage. The "sulog" file contains a log of users who execute the su(1) command. Quest provides a tool to monitor this file. It is helpful in identifying users who are trying unsuccessfully to gain access to another user's login ID (including root's) and users who have successfully gained unauthorized access to root's login. Monitoring this log may help identify unauthorized attempts to gain privileged access to the system.

Trojan Horses. Planting Trojan horses is an effective way to gain passwords and privileged access.[3] A Trojan horse is a piece of code that does some malicious activity, which the victim is usually unaware of, in addition to its regular function. It is planted in a directory where it will be executed instead of the intended program, either replacing the intended program or occurring before the intended program in the user's PATH. When the user tries to execute the intended program, the Trojan horse program is executed instead.

Examples of Trojan horses are:

- The modified who command in Example 1 in Panel 1
- A login command that not only logs in the victim but sends the login ID and password to the intruder as well
- An su command that mails the password typed by the user to the intruder.

Quest identifies all executables that reside in users' directories and have the same names as the system commands. This gives the system administrator a good start in identifying and correcting potential problems.

User .profile, umask, and PATH. In addition to checking items that the system administrator controls (such as permissions of system files and routines), Quest also checks the following items the unprivileged user controls:[5]

- The .profile. Quest checks users' .profiles and identifies any that have insecure write permissions, since a .profile that is writable by everyone permits intruders to plant Trojan horses in it.
- umask. The UNIX operating system allows users to specify a default creation mode for their files and directories. This feature is called the umask. Quest checks users' umask settings to determine if they are secure, since an intruder can take advantage of an insecure file as previously shown. An example of an insecure umask is 000, which creates files with the following permissions:

```
executable file:          rwxrwxrwx
non-executable file:      rw-rw-rw-
directory:                rwxrwxrwx
```

  If a user does not manually change permissions of files and directories after creation, the above umask will give everyone the ability to read, write, and execute all files in the user's directories. This would let an intruder plant Trojan horses, steal information, modify proprietary data, and so forth.
- PATH. Quest checks permissions of the directories listed in users' paths. Some users' PATH variables contain their own directories or their current directory before the common system bins (usually /bin and /usr/bin). If the directory is writable, this gives someone an opportunity to plant a Trojan horse.

Unprivileged users also must do their part to maintain the integrity of their systems. They can either help keep a system secure or help an intruder compromise it. Therefore, it is essential that the system administrator periodically check users' .profiles, PATHs, and umask settings and notify them if a problem is detected—actions that can be done easily and automatically with Quest.

Needs Met by Quest. All the above-mentioned security issues deal with oversights that allow a system to be compromised. Unauthorized persons can take advantage of such oversights to gain access to a system, steal information, damage system files, and remain on a system undetected. By running Quest on a regular basis, system administrators can correct many of these problems on their systems and thereby keep out intruders. Before Quest can be used to its full potential, however, some concerns must be addressed.

Current Work

The transformation of Quest during the last two years from an experimental prototype to a releasable package was shaped by a set of design rules that included modularity, portability, and speed. As feedback from Quest users has started to accumulate and the package has gained wider acceptance, issues concerning greater flexibility, user friendliness, and functionality have been raised. Each new Quest release attempted to address the problems that were most urgent at the time. Currently, because Quest is widely used throughout AT&T and on many types of UNIX systems, the directions for improvement described in the next few sections have become dominant.

Wading through the Output. Quest components, especially on large machines, can produce reports of intimidating size. This decreases the chance that the system administrator will find a serious problem amid relatively harmless warnings. The difficulty can be alleviated in two ways. Quest can make a judgment on the severity of each warning, thus giving the administrator a chance to review the most serious problems first. For example, an insecure file residing in a secure directory is, typically, a less severe problem than one residing in an insecure directory; a user's `umask` of 000 is more of a threat than a `umask` of 664; a user's `.profile` that is writable by everyone deserves more attention than a `.profile` that is merely readable by everyone; a setuid `root` file writable by everyone should arouse greater concern than a setuid `uucp` file writable by user `bin`.

Another approach, which can be used in parallel with the first one, involves greater emphasis on and customization of the exclusion files used by Quest (e.g., making publicly writable directories exempt from inspection by the component that checks write permissions).

File System Consistency. Several concerns that Quest must deal with revolve around the issue of UNIX file system consistency. These concerns are:

- Robustness in the face of file system corruption. This means that Quest must handle gracefully any contingencies arising during traversal, such as loops in the file system or directories with no parent (..) entry.
- Thoroughness of security checks. Quest was designed with the assumption that the file system is free of problems targeted by the file system-consistency-maintenance tools `fsck(1M)` and `ncheck(1M)`. Problems not reported by these commands (e.g., a .. link that does not point to the actual parent directory) should be diagnosed.
- Need for speed. Quest must be able to service UNIX system mainframes with multiple file systems. Currently, it requires many hours to run on large machines.

An approach that solves the first two concerns involves using a more thorough tree traversal. Unfortunately, that approach clashes with the need to solve the third concern. The approach currently being taken involves checking the internal representations of the directories on the device (i.e., the *inodes*) as well as the blocks containing the directories. This avoids the need to open and close each directory and issue a system call for each file in the directory, and thus consumes less time. In addition, a more thorough audit is performed because the file consistency problems listed above are handled.

Additional Features. Additional areas requiring improvement include the need to provide an interface to make Quest more user-friendly, a component that will confirm the correct installation of system security improvements, a component to cross-check system logs, and a component to check network logs and associated files.

Comparison with Other Tools

The increase in UNIX system security awareness led to the emergence of several other security auditing packages developed independently within AT&T. Among the most notable are:

- UXA
- Alert/Inform
- Sfind.

Quest, UXA, and Alert/Inform adopt a largely similar approach but differ mainly in scope, with Quest having the edge as the most widely used and functionally comprehensive package. Sfind concentrates mainly on locating file system inconsistencies and detecting setuid/setgid files.

All these tools are largely similar and are effective in lowering the risk of intrusions. However, they all suffer from a common shortcoming, described in the next section.

Future Work

For all the help that Quest can provide in identifying security problems, it has some room for growth. The possibilities are dictated by:

- Its nature as a preventive and auditing tool. While Quest will identify some suspicious records and files resulting from a (possibly successful) intrusion attempt, it will not be much help in stopping a break-in already in progress. To do so would require an on-line intrusion-detection system.[6] A system of that sort could make use of the rules for acquiring knowledge of the expected user behavior, rules for what to consider a threatening deviation from normal behavior profiles, and finally, records of user activity known as "user profiles."
- The scope of system log files. The UNIX system collects system activity records, but such a collection arrangement has not been used for security purposes.[7] Accounting is the main purpose of most UNIX System V user and process activity collection. Records of access to system *objects* (files, devices, etc.) that would allow the identification of unauthorized or denied access are lacking.

This implies that for greater effectiveness in dealing with both intrusion prevention and detection, a package such as Quest should be combined with an on-line intrusion-detection system making use of user activity records.

Summary

Quest is a software package developed to help system administrators detect problems caused by improper use of the built-in security features provided by the UNIX operating system. It has the potential to become the security audit tool for AT&T but needs some enhancements. In the meantime, Quest can be used with other tools and a set of system administration security guidelines to help keep systems more secure.

Quest is currently used on approximately 600 machines at various AT&T locations. It is still under development, with emphasis on incorporating features from the other existing AT&T security tools. The release under development will provide a component that checks for file system inconsistencies, make components tolerant of file system anomalies, assign severity categories to Quest warnings, and provide a user interface for Quest.

Acknowledgments

We wish to give special acknowledgment to the following people who are primarily responsible for the initial development of Quest: Fred Grampp, whose own security package was the basis for Quest; Bob Chancer, who worked on the initial design of Quest; and Michele Casterlin, who did most of the software development of Quest. Their significant contributions helped make Quest one of the favored security packages in AT&T. Thanks also to Jerry Vogel for his help in writing this report.

References

1. D. M. Ritchie, "On the Security of UNIX," *UNIX Programmer's Manual*, seventh edition, Vol. 2b, Sec. 2, AT&T Bell Laboratories, Murray Hill, New Jersey, January 1979.
2. R. Morris and K. Thompson, "Password Security: A Case History," *Communications of the Association for Computing Machines*, Vol. 22, No. 11, November 1979, pp. 594-597.
3. F. T. Grampp and R. H. Morris, "UNIX Operating System Security," *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Part 2, October 1984, pp. 1654-1661.
4. H. Kluepfel, "Security in a UNIX Environment," EDP Auditors Association, New York, February 1987, pp. 44-47.
5. P. H. Wood and S. G. Kochan, *UNIX System Security*, Hayden Publishing, Hasbrouck Heights, New Jersey, 1985, pp. 19-43.
6. D. E. Denning and P. G. Neumann, "Requirements and Model for IDES—A Real-Time Intrusion Detection System," Technical Report, SRI International, Computer Science Laboratory, Palo Alto, California, 1985.
7. J. Picciotto, "The Design of an Effective Auditing Subsystem," *IEEE Symposium on Security and Privacy*, Oakland, California, April 1987.

(Manuscript received February 25, 1988)