AUDITING FILES ON A NETWORK OF UNIX MACHINES

Matt Bishop, Dartmouth College

(Work reported here was supported by NASA under contract NCC 2-387 and was done at the NASA Ames Research Center, Moffett Field, CA.)

The Numerical Aerodynamic Simulator project runs a variety of UNIX based operating system on its computers (a Cray 2, 2 Amdahl 5840s, 4 VAX-11/780s, and 25 IRIS 3500 workstations, all connected by a local area network and connected to a number of wide area networks such as ARPAnet, BARRnet, and various others. Within this environment, much development is done on each machine, particularly by engineers who come from outside Ames. They are not always aware of (or respectful towards) the policies of computer security the NAS Project has set up. Worse, given the networks to which Ames is connected, an attacker who could subvert the network controls and break security could leave traces in the form of altering files in system areas (for example, to make gaining access to the system a second time easier.) For these reasons, we decided to establish a file tree auditing system.

The audit system works as follows. It scans a file system, listing name, type, protection mode, number of (hard) links, user, group, and time of last modification. The results are saved in a file, and this file is then compared to a file with the same format but containing a snapshot of expected results. Any differences are mailed to the appropriate people; they must take action to determine what to do.

The audit system is stored in its own subtree and contains several files and subdirectories. The file "Environ" contains the location of the programs the auditor uses (namely, "lstat", which generated the listing for each file; "auditls", which collates the listings for the file system; "egrep", "ls", "find", and "test", the UNIX system utilities.) The file "List" lists the roots of the file trees to be audited, and for each specify a set of options to the audit program; these options are applied only to that file tree. Master files reside here too, and are named by deleting all "/" characters from the name of the root of the file tree, and prefixing the letter "F". (If only setuid files are to be audited, the prefix is "FU"; if only setgid files are to be audited, the prefix is "FG"; and if both types of files are to be audited, the prefix is "FB".) Also here are ignore files; these files are named the same as the corresponding master files (but the "F" is replaced by an "I".) These files contain regular expressions that are used to eliminate uninteresting files.

When we had a system with which we were satisfied running on one machine, we expanded it to run multi-machine audits. This required reorganizing the program and the file structure in the audit subtree. We decided to run the audits in a master-slave relationship; the master would issue a command to the remote host to execute a program (actually, its version of "auditls") and send the output to the requester. This required two programs, "auditls" and "lstat", to be available on the remote host, so we updated the installation procedure to do this. We also had to define the mechanism to execute commands remotely; since the System V based machines used a different command than the 4.2 BSD based machines, we made this an installation time parameter. We also put the "Environ", "List", master, and ignore files for each machine into a separate directory, and created an "Equiv" file to map host names to one another, so (for example) the same machine could be referred to as "icarus" or "icarus.riacs.edu".

We quickly discovered two problems with running audits remotely. Both came about because some portions of the network software being developed were unreliable. Either the network would hang, leaving the connection alive and hung, or the network connection would be broken before the results of the remote file system scan had been completed. In the first case, the auditing process would be stopped dead in its tracks; in the second, a very large number of files would show up as being deleted, and then show up again the next day as having been created!

We dealt with both problems by making allowances for them in software. For the first, we wrote a timeout routine that executes a command, waits for a user-specified time, and then (if the process is still active) kills it and reports the termination. There is a danger that this might prematurely terminate remote file system scans running on slow or heavily loaded machines; but the timeout was set to 1 hour, and that proved to be sufficient to kill only hung processes. For the second, we made the assumption that the file systems and directories being audited changed in small increments only. So, we added a "threshhold" parameter which took action if the number of files in the remote file system were under a certain percentage of the number of files supposed to be there. For example, if the auditing system reported that directory /bin on machine chewy had 60% of the files it was supposed to have, the results of the file system scan would be saved somewhere, and a message put in the results of the audit. The message reads: "There is a potential problem with the file system /bin on chewy -- the audit showed that file system has 60% of the files it had when the master was made. Either the audit failed or most files on that file system have been deleted. Check to be sure it is not the latter, and if the master file must be regenerated, delete the current one and replace it with results.bin. Note: the master files have not been updated."

Current experience proclaims this system a success. Since the addition of the features handling the two problems described above, there have been no errors in the file audits that have not been flagged as potential errors. It has caught numerous cases where developers made private copies of privileged programs and disabled their security features. The system has been in use for about a year, and has paid off handsomely.