1998-00-00

An Intelligent Tutor for Intrusion Detection on Computer Systems

Rowe, Neil C.

Computers and Education

Computers and Education, pp. 395-404, 1998

An Intelligent Tutor for Intrusion Detection on Computer Systems

Neil C. Rowe and Sandra Schiavo

Code CS/Rp, Department of Computer Science

Naval Postgraduate School
Monterey, CA USA 93943

Abstract

Intrusion detection is the process of identifying unauthorized usage of a computer system. It an important skill for computer-system administrators. It is difficult to learn on the job because it is needed only occasionally but can be critical. We describe a tutor incorporating two programs. The first program uses artificial-intelligence planning methods to generate realistic audit files reporting actions of a variety of simulated users (including intruders) of a Unix computer system. The second program simulates the system afterwards, and asks the student to inspect the audit and fix the problems caused by the intruders. This program uses intrusion-recognition rules to itself infer the problems, planning methods to figure how best to fix them, plan-inference methods to track student actions, and tutoring rules to tutor intelligently. Experiments show that students using the tutor learn a significant amount in a short time.

This work was supported by the U. S. Naval Postgraduate School under funds provided by the Chief for Naval Operations. Special thanks to Chris Roberts and other students for debugging help. An improved version of this paper appeared in *Computers and Education*, volume 31 (1998), 395-404.

Introduction

Recent improvements in the degree of interconnection of computer systems have increased the number of threats to their security. Malicious mischief and sabotage have increased as new opportunities have arisen. Break-in threats are no longer just adolescents but can be spies and saboteurs. Good security training for system administrators is increasingly important.

Some automated tools for detection of unauthorized computer-system use ("intrusions") are now available [Lunt, 1993], of which NIDES [Lunt et al, 1989] is a good example. These tools examine records of events in a computer system ("audit files") or monitor events in real time. They do both "anomaly detection" (of statistically unusual behavior like 3 A.M. logins) and "misuse detection" (of suspicious event sequences like finding the password file and copying it). However, such automated tools require significant time and space resources and are not feasible on small computers. Also, no tool can be perfect since new threats constantly emerge and intruders learn to disguise old threats.

So people that run computer systems ("system administrators") still need training in intrusion detection. Since intrusions are unpredictable and not especially common, practice with simulators is valuable. Simulators can efficiently drill students on rare, dangerous, and/or complex intrusion problems, analogously to training of pilots with flight simulators. This paper reports on a tutor in which one simulator generates realistic audit files and another simulates the computer system afterwards while tutoring the repair of problems. The tutor supplements classroom instruction on computer security by requiring students to put security ideas together in a useful way. The tutor is implemented in Quintus Prolog and runs on Sun Sparcstations.

Our tutor teaches some anomaly detection but emphasizes misuse detection. This is because misuse detection is harder to automate since it requires applying many rules (as in NIDES) or searching for many patterns (as in [Shieh and Gligor, 1991] and [Kumar and Spafford, 1994]); anomaly detection just requires calculating statistics and comparing them to norms. Our tutor also is designed to teach Unix system security only. Intrusion detection is hard to teach without examining specific command sequences in a particular operating system, and considerable written material is available on Unix intrusion-detection procedures (e.g. [Farrow, 1991]).

Generation of simulated audits

Breadth of knowledge of intrusion methods is important to a system administrator. So a good tutor must provide a wide variety of test scenarios. Obtaining this variety through hand-coding would

be difficult and tedious, so we wrote a program to generate audit files from basic principles. Simple such programs are starting to be used for testing intrusion-detection systems [Puketza et al, 1994]. Our program Makeaudit uses artificial-intelligence planning methods to generate plans of action for each legitimate user and each intruder, maps the plans to Unix commands (allowing that some commands fail and must be repeated or abandoned), assigns realistic times to actions, generates audit records, and sorts them by increasing time.

Our planning methods are goal-directed ones derived from means-ends analysis [Newell and Simon, 1972] and extended with additional features. Most of the implementation has been used in over forty tutors as part of the Metutor system [Rowe and Galvin, 1998]. Our extensions to means-ends analysis include variables as arguments to actions, automatic avoidance of infinite loops, automatic avoidance of unachievable preconditions, time limits on subproblems, automatic backtracking on failure of initial choices, caching of results on subproblems, special debugging facilities, and automatic mapping of state descriptions to color graphics displayed in designated places on the computer screen. Action specifications are highly modular so new actions can be added easily.

To illustrate the goal specifications that generate plans, below in Prolog syntax is the goal clause for a simulated intruder. It says that (though not necessarily in this order) the intruder must determine the system administrator's password, must view the "root" and "bin" directories, must modify the executable files for the "ls" and "cd" commands, must create a file "please_run_me" that could well contain a virus, and must eventually be logged out. (Brackets delineate Prolog lists and commas delineate list items; capitalized words are variables.)

```
goal([hacked(su), viewed(root), viewed(bin),
     modified(ls,bin), modified(cd,bin),
     created(please_run_me,bin,root,22914), not(logged_in(User))]).
```

We also model normal users to challenge students to distinguish normal from suspicious behavior. Realism demands more normal users than intruders, but normal users are generally simpler to model. For example, one normal user might have the goals to view directory "smith", to create files "tmp1434" and "tmp1435" in it, to complain about any damage they notice, and to be eventually logged out.

Formal action definition requires recommending conditions, preconditions, and postconditions for the possible actions. Recommending conditions indicate action priorities by their order (the most important are first). Preconditions and postconditions may depend on the truth of other conditions. Postconditions can be random, to model actions like guessing passwords (which rarely succeeds) and inspecting files for damage (which can miss details). Here for example are the specifications of the "ls" or directory-viewing action:

1. An indirect argument to the "ls" command is the directory to which the user is currently connected.
2. A user should do "ls" in directory D if they want to view directory D.
3. The user who is not a system administrator must be logged in and at directory D to do "ls" there if directory D permits read access to everyone.
4. The user who is not a system administrator must be logged in as the owner of directory D and at D to do "ls" there if directory D permits read access only to its owner.
5. A system administrator must only be at directory D to do "ls" there.
6. Nothing becomes false when you do "ls".
7. If the user does "ls" and an intruder has tampered with the executable implementing the "ls" command, the user has seen the current directory D and has a complaint.
8. Otherwise, if the user just has seen the current directory D.
9. 30% of the time a user doing "ls" immediately forgets what they see.
10. The average time between "ls" and the next user command is 20 seconds.

Here is the Prolog form of the preceding specifications (with the underscore symbol representing irrelevant values):

```
translate_operator(ls(D),State,ls).
recommended([viewed(D)], [], ls(D)).
precondition(ls(D),
   [not(hacked(su)),file(File,D,User,_,_,_,_),file(D,_,_,_,_,[_,_,r,_],_)],
   [logged_in(_),current_directory(D)]).
precondition(ls(D),
   [not(hacked(su)),file(File,D,User,_,_,_,_),file(D,_,_,_,_,[r,_,_,_],_)],
   [logged_in(User),current_directory(D)]).
precondition(ls(D),[hacked(su),file(_,D,_,_,_,_,_)],
```

```
   [logged_in(_),current_directory(D)]).
deletepostcondition(ls(D),[]).
addpostcondition(ls(D), [tampered(ls,bin)], [viewed(D),complaint(ls,bin)]).
addpostcondition(ls(D), [not(tampered(ls,bin))],[viewed(D)]).
randchange(ls(D),[],viewed(D),[],0.3).
average_duration(ls(A),20).
```

Actions are assembled in a recursive top-down fashion to make a coherent plan for achieving all the specified goals while obeying the preconditions. For instance, an intruder who wants to insert a virus into an executable must achieve the preconditions of being in the directory of the executable and having write access to it. To be in the directory, the user must change directory from their login directory; to achieve that, they must log in. To obtain write access if they do not have it, they must do a "chmod" command, which generally requires that they log in as the owner of the executable. Thus chains of preconditions help find a proper sequence of actions.

Next, each action is mapped to a corresponding Unix command, and standard auditing information (name of user, directory they are in, command, and result) is calculated. Realistic times of events are found by assigning to each action a duration that is a normal random variable with parameters specific to the action, and adding this to an evenly distributed random time of user login. Results of actions are then adjusted for time-dependent effects. For instance, if a malicious user changes a file, this will only be noticed by users accessing the file later.

Here is part of an audit file generated by Makeaudit:

```
jones, 933, root, "login root", fail
jones, 945, root, "login root", fail
jones, 948, root, "login root", fail
jones, 974, root, "login root", fail
jones, 977, root, "login root", fail
jones, 988, root, "login root", fail
jones, 993, root, "login root", ok
root, 1013, root, "cd  root/etc", ok
root, 1019, etc, "cp passwd  smith/dont_read", ok
root, 1209, etc, "mail root", "Captain Flash strikes again!!!!"
root, 1216, etc, logout, ok
davis, 3801, davis, "emacs goodnews", 1437
davis, 3809, davis, logout, ok
adams, 5369, adams, "cd  root/bin", ok
adams, 5388, bin, ls, ok
adams, 5394, bin, "cd  adams", ok
adams, 5395, adams, "cd  dog", ok
adams, 5403, dog, ls, ok
adams, 5413, dog, "cd  tom", ok
adams, 5442, tom, ls, ok
adams, 5458, tom, "cd  adams", ok
adams, 5463, bin, "cd  uri", ok
adams, 5463, uri, ls, fail
uri, 5487, farmer, "cat secrets", ok
adams, 5493, uri, ls, fail
uri, 5493, farmer, logout, ok
adams, 5530, uri, ls, fail
adams, 5573, uri, ls, ok
adams, 5589, uri, "cd  adams", ok
adams, 5596, adams, "cd  tom", ok
```

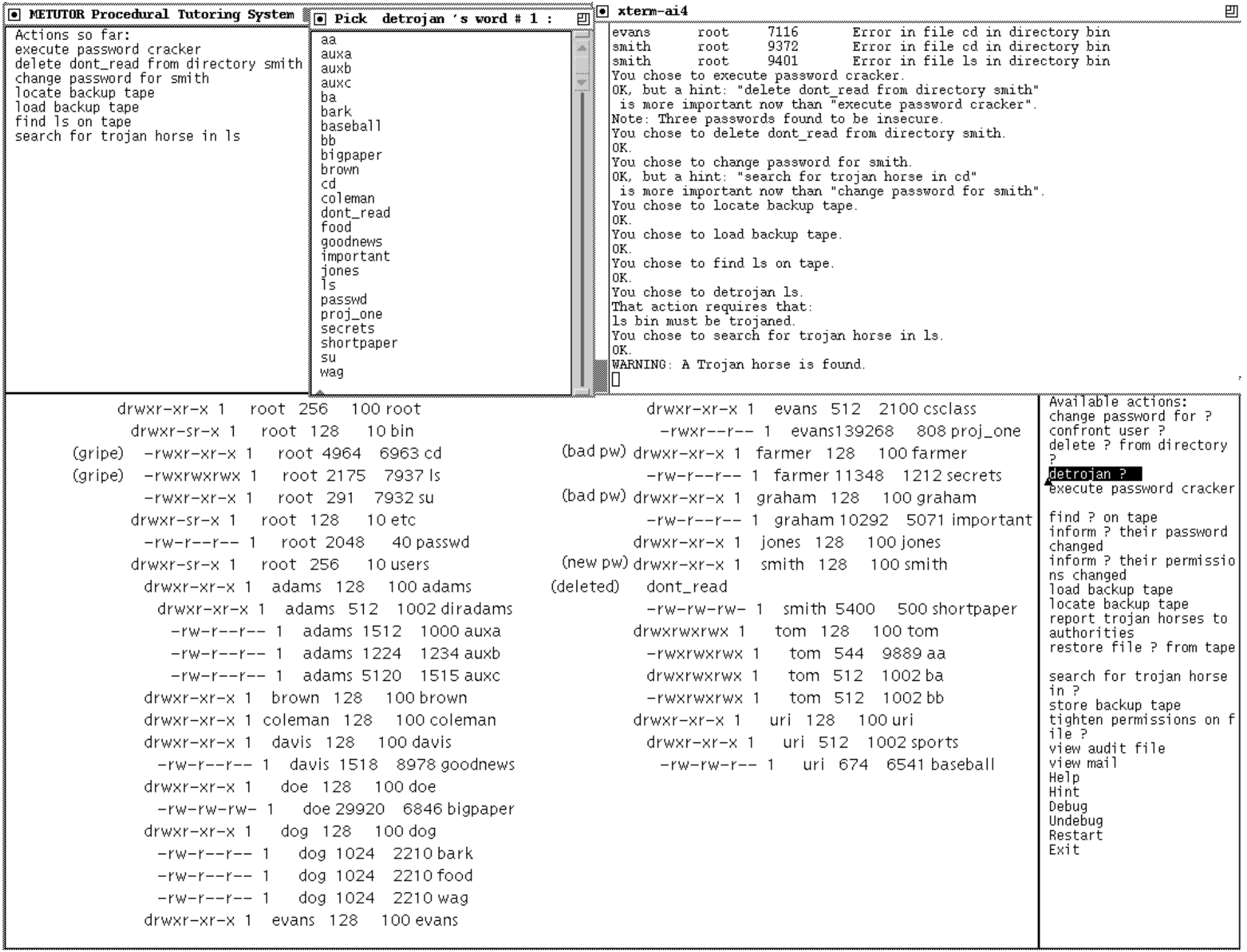
```
adams, 5647, tom, "rm *", ok
adams, 5760, tom, "mail tom", "Haha ful"
adams, 5765, tom, logout, ok
smith, 5826, none, "login smith", ok
smith, 5854, smith, ls, ok
brown, 6091, tom, "emacs ba", 628
smith, 6237, smith, "emacs tmp1434", 344
brown, 6256, tom, "mail root", bad(ba, tom)
brown, 6263, tom, logout, ok
```

The first item on each line is the user name; the second the time in seconds since start of the audit; the third the directory the user is in; the fourth the Unix command they gave; and the fifth the additional necessary information (like whether the command failed or the size of a created file). Here four normal users Davis, Uri, Smith, and Brown pursue normal activities. Meanwhile an intruder masquerading as Jones guessed the system-administrator ("root") password, copied the password file, and sent a taunting message to the real system administrator. A second intruder masquerading as Adams (simulating a high-school student with poor spelling) examined a number of directories before deleting all the files in Tom's directory and taunting him. Later, user Brown noticed that Tom's file "ba" has been damaged by an intruder, and sent a complaint to the system administrator.

Intelligent tutoring methods

The student's job is to study the audit file and fix the observed problems as a system administrator would. Typically, this requires several steps. For instance, if an intruder destroys some of Tom's files, then the student must find the destroyed files on backup tape and restore them. The student should also check Tom's password to see if it was easy to guess; if so, a message should be sent to Tom telling him what happened and how to get the new password. The student should also check the protections (discretionary access specification) on the destroyed files to see if they enabled unfairly easy access. Passwords and protections should be checked first to prevent further damage. Related tasks are checking complaints from other users about Tom's files, and comparing online and backup copies of Tom's remaining files to find viruses. If the destroyed files were executables, files created or modified by the executable may also be tainted and should be checked too.

The Idtutor tutor observes this process like a coach. It shows the state of the file system, offers intelligent hints (like [Burton and Brown, 1982]), and stops the student only when they try to do something impossible. Figure 1 shows an example snapshot of the user interface.


METUTOR Procedural Tutoring System
Actions so far:
execute password cracker
delete dont_read from directory smith
change password for smith
locate backup tape
load backup tape
find ls on tape
search for trojan horse in ls
Pick detrojan 's word # 1 :
aa
auxa
auxb
auxc
ba
bark
baseball
bb
bigpaper
brown
cd
coleman
dont_read
food
goodnews
important
jones
ls
passwd
proj_one
secrets
shortpaper
su
wag
xterm-ai4
evans root 7116 Error in file cd in directory bin
smith root 9372 Error in file cd in directory bin
smith root 9401 Error in file ls in directory bin
You chose to execute password cracker.
OK, but a hint: "delete dont_read from directory smith"
is more important now than "execute password cracker".
Note: Three passwords found to be insecure.
You chose to delete dont_read from directory smith.
OK.
You chose to change password for smith.
OK, but a hint: "search for trojan horse in cd"
is more important now than "change password for smith".
You chose to locate backup tape.
OK.
You chose to load backup tape.
OK.
You chose to find ls on tape.
OK.
You chose to detrojan ls.
That action requires that:
ls bin must be trojaned.
You chose to search for trojan horse in ls.
OK.
WARNING: A Trojan horse is found.
drwxr-xr-x 1 root 256 100 root
drwxr-sr-x 1 root 128 10 bin
(gripe) -rwxr-xr-x 1 root 4964 6963 cd
(gripe) -rwxrwxrwx 1 root 2175 7937 ls
-rwxr-xr-x 1 root 291 7932 su
drwxr-sr-x 1 root 128 10 etc
-rw-r--r-- 1 root 2048 40 passwd
drwxr-sr-x 1 root 256 10 users
drwxr-xr-x 1 adams 128 100 adams
drwxr-xr-x 1 adams 512 1002 diradams
-rw-r--r-- 1 adams 1512 1000 auxa
-rw-r--r-- 1 adams 1224 1234 auxb
-rw-r--r-- 1 adams 5120 1515 auxc
drwxr-xr-x 1 brown 128 100 brown
drwxr-xr-x 1 coleman 128 100 coleman
drwxr-xr-x 1 davis 128 100 davis
-rw-r--r-- 1 davis 1518 8978 goodnews
drwxr-xr-x 1 doe 128 100 doe
-rw-rw-rw- 1 doe 29920 6846 bigpaper
drwxr-xr-x 1 dog 128 100 dog
-rw-r--r-- 1 dog 1024 2210 bark
-rw-r--r-- 1 dog 1024 2210 food
-rw-r--r-- 1 dog 1024 2210 wag
drwxr-xr-x 1 evans 128 100 evans
drwxr-xr-x 1 evans 512 2100 csclass
-rwxr--r-- 1 evans139268 808 proj_one
(bad pw) drwxr-xr-x 1 farmer 128 100 farmer
-rw-r--r-- 1 farmer 11348 1212 secrets
(bad pw) drwxr-xr-x 1 graham 128 100 graham
-rw-r--r-- 1 graham 10292 5071 important
drwxr-xr-x 1 jones 128 100 jones
(new pw) drwxr-xr-x 1 smith 128 100 smith
(deleted) dont_read
-rw-rw-rw- 1 smith 5400 500 shortpaper
drwxrwxrwx 1 tom 128 100 tom
-rwxrwxrwx 1 tom 544 9889 aa
drwxrwxrwx 1 tom 512 1002 ba
-rwxrwxrwx 1 tom 512 1002 bb
drwxr-xr-x 1 uri 128 100 uri
drwxr-xr-x 1 uri 512 1002 sports
-rw-rw-r-- 1 uri 674 6541 baseball
Available actions:
change password for ?
confront user ?
delete ? from directory ?
detrojan ?
execute password cracker
find ? on tape
inform ? their password changed
inform ? their permissions changed
load backup tape
locate backup tape
report trojan horses to authorities
restore file ? from tape
search for trojan horse in ?
store backup tape
tighten permissions on file ?
view audit file
view mail
Help
Hint
Debug
Undebug
Restart
Exit

The lower left is a graphical view of the current file hierarchy, the lower right is the menu of possible actions for the student to select with the mouse, the upper right holds informational messages for the user, the upper middle is a popup window prompting for a file name, and the upper left shows the actions taken so far. The lower left display is Unix "ls -l" command output enhanced with indentations, notations about deleted files, notations about received complaints, and notations for accounts with poor ("bad pw") or new ("new pw") passwords. The menu of actions includes review of task objectives (under "Help"), review of the audit, review of the mail, debugging control, and receiving a hint in the form of a suggested action (obtained by real-time planning with the current state and the original goals). Fig. 1 shows the moment when the user has just chosen to "detrojan" but has not yet selected the file (file "ls" would make sense to pick as a Trojan horse was just found there). As seen in the upper left, this will be the eighth action of the student. As seen in the upper right, the tutor grumbled about the first and third actions, and refused to "detrojan" earlier when no Trojan horse had been found. Also note the end of a summary of incoming mail in the first three lines of the upper right, showing that users Evans and Smith complained about file "cd" and user Smith about file "ls".

Idtutor is built on top of the Metutor shell for tutoring of precisely-defined procedural skills [Rowe and Galvin, 1998]. Metutor is similar in philosophy to the expert-system shell approach of [Sleeman, 1987] for specifying tutoring systems, but differs considerably from the popular decision-graph approach to teaching procedural skills of (e.g. [Johnson, 1987]). Idtutor uses similar planning methods to those of Makeaudit, but applied to remediation goals and remediation actions (not to be confused with planning for active instructional management as in [Peachey and McCalla, 1986]).

For instance, here are the specifications for restoring a file F from tape backup:

1. A user should restore file F from backup if they want F to no longer be deleted or modified.
2. A user can restore file F from backup if they are in the directory where F should go, F has a safe backup copy, they have found the copy on the backup tape, and the person who damaged F is not currently logged in.
3. When a file is restored, it is no longer deleted or modified, and it no longer has a "tamperer".
4. Restoring a file fails 10% of the time with the message "Backup damaged too".

Here is the Prolog form of the above:

```
recommended([not(deleted(F))], restore_from_backup(F).
recommended([not(modified(F))], restore_from_backup(F).
precondition(restore_from_backup(F),
   [file(F), current_directory(D), directory_of(F,D), safe_backup(F),
     found(F,on,tape),tamperer(F,P), not(logged_in(P))]).
deletepostcondition(restore_from_backup(F),
   [deleted(F), modified(F), tamperer(F,P)]).
addpostcondition(restore_from_backup(F), []).
randchange(restore_from_backup(F), [], [deleted(F)], 0.1, 'Backup damaged too.').
```

Action specifications like these are modular, permitting easy addition of new countermeasures for intrusions. Specification of the meaning and purpose of every remedial action permits Idtutor to plan complete remediation sequences from whatever state in which the student finds themselves. This permits Idtutor to give hints tailored to the student's exact circumstances rather than on general principles. It also allows Idtutor to recognize and provide focused tutoring on useless, dangerous, digressing, and low-priority actions, and permits inference of the student's syntactic confusions (between actions with similar names) and semantic confusions (between actions of similar effects). Tutoring is managed by Metutor's domain-independent rules for tutoring of procedural skills. While a traditional authoring approach with a structured sequence of questions [Steinberg, 1991] could tutor somewhat similarly, using the Metutor approach meant faster construction of a thorough tutor, much more compact representation of it, and greater ease of modification, as discussed in [Rowe and Galvin, 1998].

To remediate intrusion problems, both the student and tutor must recognize suspicious behavior in the audit record. The tutor does this with common-sense rules. Since most these are obvious, students are tutored about them only indirectly in comments on their remediation actions. Example rules are:

1. If two people are logged in at the same time under the same login name, suspect that one of them is an intruder;
2. If more than one file is changed by the same number of bytes by a user in a session, suspect planting of viruses or Trojan horses;
3. If the system administrator gets mail complaining about a file, and the file has been edited recently, suspect malicious modification of it.

Data summarizing Idtutor sessions is recorded for later inspection by an instructor. This data includes the student name, the exercise, the duration of the session, the CPU time used, the number of student errors and hints, and a full listing of the errors and hints with their context.

Experiments

We tested our tutor on 19 randomly selected students in an introductory computer security course; 19 additional students served as a control group. The students were in M.S. degree programs, and many of them will manage computer facilities after they leave our school. The students heard introductory lectures on intrusions and their remediation. We gave them three exercises, each with a different audit file of about 100 lines in length referring to a varied set of 40 starting files and executables.

Students had to work on each exercise until they had fixed every simulated problem, which required about 20 actions for each exercise. The first exercise was unmonitored so students could practice. The other two exercises took an average of 23.6 minutes each to solve (with a standard deviation of 14.8 minutes). The exercises each required an average of 0.038 minutes of CPU time, so the tutor does not appear to require much processing-time overhead despite its intelligent reasoning methods. Students averaged 7.79 errors per run, if we call any suboptimal action an "error", although most of these were subtle. Students asked for a hint (about the action that the tutor prefers) an average of 1.7 times per exercise.

Learning was assessed by a post-test of eight multiple-choice questions on general features of intrusion detection and remediation, questions deliberately focusing on underlying principles not explicitly taught by the tutor. For example, "Intrusion detection is hard because: (a) there are many and varied threats; (b) most threats are difficult to see; (c) you need a Ph.D. to do it adequately; (d) there's considerable disagreement over what is an 'intrusion.'" Students who used the tutor got an average of 5.94 questions of 8 correct (with a standard deviation of 1.08), compared to 4.74 of 8 (with a standard deviation of 1.48) for the control group of 19 students who received just the lectures on intrusion detection. This difference is significant to the 99.6% level using Fisher's 2-by-2 test. Since the students were selected randomly for test and control groups, use of the tutor appears to explain the difference in performance, though it might also be due to the effect of the tutor's novelty. Post-tests also confirmed that students preferred the tutor to an equivalent amount of classroom-lecture time. Thus the tutor seems effective.

References

R. Burton and J. Brown, An investigation of computer coaching for informal learning activities. In *Intelligent Tutoring Systems*, ed. D. Sleeman and J. Brown, London: Academic Press, 1982, pp. 79-98.

R. Farrow, *Unix System Security*. Reading, MA: Addison-Wesley, 1991.

W. Johnson, Developing expert system knowledge bases in technical training environments. In *Intelligent Tutoring Systems: Lessons Learned*, pp. 21-33. New York: Lawrence Erlbaum, 1987.

S. Kumar and E. Spafford, A pattern-matching model for misuse intrusion detection. Seventeenth National Computer Security Conference, Baltimore, MD, October 1994, 11-21.

T. Lunt, A survey of intrusion-detection techniques. *Computers and Security, 12*, 4 (June 1993), 405-418.

T. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, and S. Listgarten. Knowledge-based intrusion detection. Conference on AI Systems in Government, March 1989, Washington DC, 1-6.

A. Newell and H. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

D. Peachey and G. McCalla, Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies, 24* (1986), pp. 77-98.

N. Puketza, B. Mukherjee, R. Olsson, and K. Zhang, Testing intrusion detection systems: design methodologies and results from an early prototype. Seventeenth National Computer Security Conference, Baltimore, MD, October 1994, 1-10.

N. Rowe and T. Galvin, An authoring system for intelligent tutors for procedural skills. *IEEE Intelligent Systems, 13*, 3 (May/June 1998), 61-69.

S. Shieh and V. Gligor, A pattern-oriented intrusion-detection model and its applications. Symposium on Security and Privacy, Oakland, CA, May 1991, 327-342.

D. Sleeman, PIXIE: A shell for developing intelligent tutoring systems. In *Artificial Intelligence and Education: Volume I*, ed. R. Lawler and M. Yazdani, 239-265. Norwood, NJ: Ablex, 1987.

E. Steinberg, *Teaching Computers To Teach, second edition*. Hillsdale, NJ: Lawrence Erlbaum, 1991.

Go to paper index