Edgar H. Sibley
Panel Editor

A superset of the secondary passwords technique, pass-algorithms provide enormous flexibility and ease of implementation and represent an attractive alternative to more costly systems security features and equipment. The method validates users who demonstrate knowledge of a secret algorithm rather than a password.

PASS-ALGORITHMS: A USER VALIDATION SCHEME BASED ON KNOWLEDGE OF SECRET ALGORITHMS

JAMES A. HASKETT

Over the last few months, there has been a surge of interest in both the public and trade press about a number of computer break-ins. On the whole, these break-ins were initiated by novices using dialup facilities and home computers to make repeated attempts at guessing user names and passwords, and were noteworthy in that they were completed without the aid of sophisticated equipment.

In any computer system, there are a number of areas where protection must be provided (e.g., physical security to bar unauthorized access to the computer console and power supplies). In addition, the system manager must take full advantage of the security features provided by the vendor, among them some form of log-in protection. Customers and users frequently implement vendor log-in-protection features inadequately and therefore do not have the protection that they might otherwise enjoy.

Among the devices and techniques available to improve log-in security are costly devices like thumb prints and retina scans. Some sites provide log-in security protection by restricting computers and terminals to a secure area delimited by guards and identification badges.

While the battle between those who would have tight security and those who want simpler (user-friendly) log-ins continues, some of us find ourselves needing an immediately implementable system that can be enforced on an as-needed basis as determined by the system manager, the security officer (or project manager, supervisor, instructor, etc.), and/or the user.

A technique known as secondary passwords is sometimes used to provide supplemental log-in security to that provided by the vendor. Secondary passwords are often easy to implement by the customer without local modifications to the operating system, but essentially provide only more of the same kind of protection. Also, many users find secondary passwords to be an extraordinary inconvenience. This paper proposes the use of pass-algorithms, a superset of the secondary passwords technique that validates users who demonstrate knowledge of a secret algorithm rather than a password.

THE PROBLEM WITH PASSWORDS

By far the most frequently used log-in-protection scheme is passwords. A user types his or her user name and password at a terminal, and the operating system compares those two items with entries in the valid-user file. If a match is found, the user is considered a valid

user and is permitted to use the system.

The standard password technique is enforced by the operating system in such a way that customers cannot tailor it without making modifications to the operating system. For many sites, this restriction makes tailoring either unacceptable or impossible.

Also, the password-verification algorithm is simplistic (the typed password, perhaps after encryption, must exactly match the password stored in the valid-user file). Finally, the password, rarely changed by users, is relatively static.

In order to deal with what the vendor supplies, system managers usually do not attempt to provide more log-in security. Instead, many simply encourage users to use long, nonsense passwords and to change them often. Nonsense passwords are encouraged so that onlookers will find it difficult to remember them, and so that intruders using programmed microcomputers with dial-up capabilities will find it unacceptably time-consuming to try all combinations of user names and passwords in the hope of finding a valid pair. The problem with long, nonsense passwords is that they are also difficult for the owner to remember.

In practice, site recommendations are often ignored: Users tend to prefer short, easily remembered passwords that are rarely changed. Names of friends, social security numbers, and names associated with personal interests are typical.

RECONSTRUCTED PASSWORDS

Several years ago, a group of us at Bloomington Academic Computing Services began to experiment with a scheme that might be called reconstructed passwords. This allowed us to continue to use the vendor's password-protection scheme without modification or addition and still make use of long, nonsense passwords. However, in this scheme, the passwords are not really nonsense—they only appear so to the onlooker. They are, in fact, reconstructed by the owner in the process of typing them and basically fall into two classes.

The first class might be represented by the musician who wants a 25-character password but finds it difficult to remember one. However, he or she can "play" 25 notes of a favorite but obscure piece on the keyboard. Likewise, a knitter might "knit and purl" 25 stitches from a favorite sweater on the keyboard. (Such a password can be particularly effective because it contains only two symbols ("k" and "p") that seemingly appear at random. Thus, there are no memory hooks for the onlooker.)

In the second class would be the sports fan who remembers the batting averages of a favorite player over the past 10 years. The fan types the averages in, say, reverse chronological order. Likewise, the designer of a microcode machine might remember which gates are open and which closed for the adder, shifter, and registers of a particularly elegant instruction. Mentally scanning the logic charts, he or she reconstructs the password, which consists of only 0s and 1s.

Although the use of reconstructed passwords does improve security and is a technique worthy of use in its own right, it still suffers from the general deficiencies of password protection as currently practiced. However, an examination of the second class of reconstructed passwords indicates some interesting possibilities. The user is no longer required to remember one single, long password but instead remembers two things—several short subpasswords and an algorithm for fitting them together. This means that now the onlooker must either remember a long, nonsense password or discover both the algorithm and all the subpasswords.

This second class of reconstructed passwords provides the first hint regarding the secondary password superset technique known as pass-algorithms. Why should the user remember the subpasswords (e.g., the batting average)? Perhaps we can let the computer "remember" them (computers are good at remembering data) and let the user remember the algorithm (people are good at remembering methods). Furthermore, why not make the password change every time the user attempts to log in or moves from one terminal to another, perhaps requiring a long log-in sequence when he or she works from a dialup terminal and a short one in a highly secure area?

PASS-ALGORITHMS

The idea of using a different password every time a user logs in may provoke some initial outrage from users and security officers who imagine users grudgingly walking around with a list of this month's passwords in their pockets. Thankfully, that is not the case. For example, after a user has successfully logged in via the vendor's standard log-in procedure, he or she suddenly finds a randomly generated prompt of BEL on the terminal. Since the system manager has told the user that this month's secret log-in algorithm requires providing the next alphabetic character for each character of the prompt, the user responds with the password CFM and completes a successful log-in.

Note that there are now two levels of security. First, as in the usual password implementation, echo of the password CFM is suppressed so that an onlooker cannot read it from the terminal. Second, even if the password were printed on the terminal, that password is valid only for this particular log-in attempt. (The algorithm does not appear on the terminal and is hence unknown to the onlooker.) This means that a successful intruder working as an onlooker must now discover an algorithm rather than a password. Knowledge of the password alone is insufficient because the prompt (BEL above) was randomly generated, and both the prompt and the password will be different the next time the user (or the intruder) attempts to log in.

Furthermore, because the password changes from one log-in attempt to the next, an intruder cannot use the rejection of a password to imply anything about the next password to be tried. That is, the rejection of AAAA does not imply that AAAA will not be the proper password at the next log-in attempt, only that

AAAA was incorrect for this attempt.

The prompt can be generated in a number of ways. It may be the day, the time, the temperature, the system real-time clock, the amount of disk space available, the company's current stock price, or the port number of the current terminal. To be particularly effective, the prompt might consist of all of these and more. The correct algorithm, however, might use only one of them. This provides a third level of security, since an intruder must now discover which part of a perhaps several-hundred-character prompt is the key.

The correct algorithm might also use none of the prompts; that is, the prompt might be a ruse. The proper response could be any password of more than 8 and less than 15 characters in length that contains none of the home keys and is entered more than 5 but less than 10 seconds after the prompt is displayed. The proper response might even be keyed to the previous prompt, with the current prompt used only as camouflage; or, the prompt might be a duplicate of the previous prompt, which would confuse an onlooker into thinking that the last password offered was incorrect and another try should be made.

It is also conceivable that the proper response to the prompt depends upon terminal location, perhaps as determined by the port number. This will not work on terminal networks where any terminal can be connected to any port at a given time. Using the next alphabetic character algorithm, if a terminal were on a dialup line, the valid response to the prompt BEL might be "wucsmfxymap." If the same prompt had been displayed at a terminal in a semisecure area and the same algorithm were in use, the valid response could be the significantly shorter "acdflmq." The same prompt displayed in a highly secure area, using the same algorithm, might invoke the very short valid response "cfm." Note that the true passwords are the same in all three cases. In this algorithm, terminal location determines the number of random confusion characters that the user must intersperse between the characters of the true password. With this method, it is easy to create long, nonsense passwords for use in unsecure areas and at the same time allow short passwords for the user working at the customary station in a secure area. As a bonus, even if an onlooker sees a user absentmindedly enter the short version (cfm) on a terminal in an unsecure area, the password is useless without the algorithm because the password is valid only in the highly secure area where he or she is not permitted.

There are many algorithms that can be used. Assembly language programmers, for example, might create algorithms using indexed, indirect, or relative addressing in the prompt string. The correct algorithms may also change on odd-numbered days.

It is important that the log-in-protection programs not provide the user with any indication that the log-in sequence has been completed. The log-in programs should continue to accept input from the user as though it were one more of the required passwords until the user enters an end-of-passwords password. Terminating the log-in-protection routines when the proper number of passwords have been entered (and letting the operating system issue the command prompt) tells the intruder that he or she has successfully broken in. Conversely, to confuse an intruder, the log-in-protection program might issue a prompt that is identical to that of the operating system. In general, the log-in programs should accept a large number of the offered passwords in an attempt to mislead the intruder and then log him or her off before using too many resources. In addition, as soon as the log-in program determines that an illegal password has been entered, an alternate but erroneous series of prompts may be issued to cause further confusion.

HOW THIS PROTECTION CAN BE ENFORCED

This type of log-in protection is easily installed on operating systems that execute a command file or program that is provided by the site each time a user logs in. Although not all operating systems provide this feature, Digital's VAX/VMS, Control Data Corporation's NOS, PR1ME's PRIMOS, and Berkeley's UNIX 2.9 BSD do. Note that this is the enabling feature for the technique described in this paper.

Providing this kind of pass-algorithm protection requires that the system manager ensure that (1) all log-in attempts, especially those to privileged accounts, execute the system log-in command file; (2) the user cannot at any time abort the execution of the system log-in-command file; and (3) the pass-algorithm-protection routines provided by the site, security officer, and user are invoked by the system log-in command file (also with user abort inhibited). (This is probably most easily done by altering the system log-in command file to look for files having a specified name in the directories of the security officer and user. See the Appendix for implementation specifics for a VAX running VMS.) Failure to take these three steps may result in an intruder's circumventing the entire additional security protection.

If either the system manager, security officer, or user wishes to implement pass-algorithms, they must write programs to prompt the user and ensure that the entered passwords meet the demands of the log-in-protection programs. Sites may even wish to create libraries of algorithms that the system manager, security officer, and user may call from high-level programs. These libraries should be large and the routines highly parameterized so that onlookers who are also legitimate users cannot guess too much about the log-in protection that a particular user has selected.

EXAMPLES

Suppose that a company has a computer that is used by its customers to list highly volatile pricing information. The company has decided that this information must be quickly and easily accessible to its customers, and hence short user names and passwords are encouraged. Furthermore, there must be no log-in-protection devices that would cause undue inconvenience to customers. At the same time, the company recognizes that

the pricing information is sensitive enough that it must be protected from unauthorized changes.

It is decided that responsibility for security enforcement on this machine will lie with the security officer. The system manager is instructed to include in the log-in command file whatever commands are necessary to execute a special program every time a user logs in. The security officer will provide this program.

The program provided by the security officer checks the user name and quickly exits if it belongs to either a customer or most in-house users. If the user name is on a list of in-house users who are permitted to alter the price list, the program requires the user to demonstrate his or her knowledge of a secret algorithm that the security officer has provided. The security officer changes the algorithm monthly and notifies only those with a need-to-know via handwritten note.

Suppose that the same company has another computer for all its development work. This work is so highly confidential that only certified users are given access. All log-ins on this machine must undergo special security checks, which the system manager is responsible for enforcing. These security checks must ensure that members of each of the two development groups cannot access user names belonging to members of the other group. In addition, a corporate officer responsible for future contracts also has sensitive files that necessitate additional protection for his or her user name. The officer is responsible for providing the additional protection that will prevent other users from logging into his or her user name. On occasion, other users might also have a need for such additional protection.

In this situation, the system manager adds the necessary commands to the system log-in command file to require users to demonstrate their knowledge of the secret algorithm used for their respective development group in a particular month. The appropriate secret algorithm is given to only those who need to know. Furthermore, the system manager adds the necessary commands to check each user's directory for a special file called SECURITY. If the user (e.g., the contracts officer) is working on particularly sensitive projects and has such a file, the system log-in command file will execute it. That program will then require the user to demonstrate knowledge of another secret algorithm that the user him- or herself has provided.

This means that, with the cooperation of the system manager, some users can force additional security on other users (e.g., the security officer on those who would alter the price list in the first example, and the system manager on all users in the second) or can use it for themselves (e.g., the contracts officer in the second example).

TRAPPING INTRUDERS

A peculiarity of secondary password protection is that, as far as the operating system is concerned, the user has successfully logged in after acceptance of the primary password and therefore has all the privileges associated with that user in his or her validation record, including permission to write and create files in the user's directory. However, in the case of pass-algorithms, the user is not validated after acceptance of the primary password and may not write to files, etc., whereas the log-in program may. In particular, the log-in program can record the date and time of the current log-in attempt, the port number through which the attempt was made, and any other information that might be available, including the offending passwords. This knowledge, particularly the guessed passwords, may aid in attempts to track down intruders. It also ensures the account owner when he or she next logs in that there have been no attempts at forced entry.

CAVEATS

Because the log-in-protection files and programs contain the algorithm that is now the key to log-in protection, they must be accessible on an execute-only basis. If they are not well protected, onlookers who are also valid users may be able to determine the algorithm from copied files, memory dumps, etc., and thereby gain access to other accounts.

If the enforcement program contains different algorithms for different users, steps must be taken to ensure that one valid user cannot discover the algorithm for other users by, for example, dumping memory immediately after log-in. If the enforcement programs use algorithms that require the user's active participation, they must not be executed for batch jobs. Since batch jobs are usually submitted by an already validated user and the accounting file can be used to trace the submission, this is probably not a serious objection to the use of pass-algorithms.

Finally, there may be a need for an escape mechanism. In a site with many valid users, systems managers could spend a great deal of time circumventing the users' own log-in-protection programs that suddenly go awry. (This is the successor to the problem of users who forget their passwords.) Users can build escapes into their own log-in programs by reaching an agreement with a fellow user. That is, user A might code his or her log-in-protection program to exit under the condition that user B has a certain publicly accessible file that contains a certain very long password. User A provides user B with the name of the file and the password to be created only if the need arises.

SUMMARY

A superset of the log-in-protection scheme known as secondary passwords, pass-algorithms are inexpensive, easy to install and maintain, and require no modifications to the operating system. In addition, the technique is operating-system independent, flexible, and allows security enforcement by any of a number of people who might be so charged. Such individuals can enforce as much validation as deemed necessary in terms of the project, terminal location, or other factors. The pass-algorithm scheme also allows enforcers to collect site-dependent information about attempted break-ins.

APPENDIX

How to enable enhanced passwords on a VAX running VMS:

1. Set /CAPTIVE in AUTHORIZE for all users.
2. Add the following to the system log-in command file:

SYS$MANAGER:SYSLOGIN.COM

```
$!Execute the site log-in protection
$!
$ IF F$MODE() .NES. "INTERACTIVE" THEN GOTO LI
$ OPEN D SYS$MANAGER:SECURITY.EXE
  /ERROR=NO_SITE
$ CLOSE D
$ RUN SYS$MANAGER:SECURITY
$ NO_SITE:
$!
$!Execute the Security Officer's log-in protection
$!
$ OPEN D SYS$SECURITY:SECURITY.EXE
  /ERROR=NO_SO
$ CLOSE D
$ RUN SYS$SECURITY:SECURITY.EXE
$ NO_SO:
$!
$!Execute the user's log-in protection
$!
$ OPEN D SECURITY.EXE /ERROR=NO_USER:
$ CLOSE D
$ RUN SECURITY.EXE
$ NO_USER:
$!
$!Enable User aborts
$ SET CONTROL=(Y,T)
$!
$!Execute the user's LOGIN.COM file
$!
$ LI:
$@LOGIN
```

CR Categories and Subject Descriptors: K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*security*
General Terms: Management, Security
Additional Key Words and Phrases: passwords, pass-algorithms

Received 1/84; accepted 3/84

Author's Present Address: James A. Haskett, Bloomington Academic Computing Services, HPER 78D, Indiana University, Bloomington, IN 47405.