Peter J. Denning

Passwords

In 1960, the year of the first time-sharing systems, we crossed a divide in the history of computing. Time-sharing systems were the first to allow multiple users interactive access to their files in the same memory system. They were the first to allow users to specify who else could read and write their files. The password was invented as a means of identifying users to the system so that access specifications could be enforced.

Sometime in the 1980s we crossed another divide. Most users no longer worked with time-sharing systems; instead they worked with personal computers that were usually linked by networks. Cryptographic protocols were invented to supplement passwords for establishing the authenticity of a user who initiated a communication between machines.

Little about the way passwords work has changed since 1960. A system administrator creates an account and associates a password with a user's name. When the user types an account name at an idle workstation, the login program that controls access requests the password but does not display what the user types. If the password matches the representation saved in the system's password file, the login program creates a session process for that user. From that moment, all attempts at file access by that process are validated by checking that the user name associated with it is stored in the file's access control list. The password is thus the key to a portion of the file system. The password of the system manager, such as "root" in a Unix system, is the key to most or all the files of the system. Thus password security has been central to the privacy and integrity of a person's files.

In the early systems, the password file was accessible only to the system administrator and to the login program. Accidents leading to the release of this file were not uncommon: Text editors usually left temporary copies in public directories, the backup system made copies on archive tapes, and forgetful

system operators left copies of the file on the line printer. By the late 1960s, the standard practice had shifted from storing passwords in plain text to storing passwords in cipher. With a one-way cipher, designers could assure that no one reading the password file could deduce the plain-text passwords. A login program simply enciphers a fixed string with the user's typed password as key, and then it compares the result with that user's entry in the password file.

Password Cracking

Even though passwords are enciphered by an irreversible function, password cracking can be easy. The attacker needs only encipher guessed passwords and compare the results with the password file. A study in 1979 by Robert Morris and Ken Thompson of AT&T Bell Laboratories demonstrated with Unix that most users choose passwords that are easily guessed -- a one- or two-letter combination, a word in the dictionary, or the person's own name (1). In the systems Morris and Thompson tested, 85 percent of the passwords were of this kind. In a matter of minutes, an attacker could try all these easy combinations and have an 85 percent chance of discovering at least one password. To make the cracker's job more difficult, Morris and Thompson added a "salt", a bit pattern that appends itself to the password (Figure 1). A 12-bit salt increases the cracker's work by a factor of 4096.

   //FIGURE 1 ABOUT HERE//

Attacks by password crackers were aided by the Unix convention of allowing the password file (named "/etc/passwd") to be readable by anyone -- a convention that simplified local authentication in early UNIX systems but became an open invitation to would-be crackers in the Internet. A cracker copied the file to a separate computer, spent as long as needed to crack it by using the heuristics above, and then logged in to the cracked account on the first try. Many Unix systems today store fake ciphers in the public password file, placing the real ciphers in a separate "shadow" file accessible only to the login program.

The 1988 Internet worm was a massive password attack (2). At the time there were approximately 60,000 computers on the Internet; the worm invaded around 3,000 of them. A large segment of the worm's code was devoted to guessing passwords in the next system to be invaded. Today, with nearly twenty million computers active on the Internet, there is an abundance of systems for would-be intruders to attack. And attack they do. From an attacker's view, finding a system with at least one easily guessed password is merely a matter of time. The Computer Emergency Response Team (CERT) at the Software Engineering Institute of Carnegie-Mellon University issues frequent advisories about systematic attacks against computers on the Internet. The vast majority of the break-ins investigated by CERT have involved the cracking of passwords.

Many system administrators now routinely set policies that would, if heeded, significantly reduce the odds of a successful password-guessing attack. A useful compendium of these policies for Unix has been assembled by Simson Garfinkel of *NextWorld* magazine and Eugene Spafford of Purdue University (3). Among their guidelines: use seven or eight characters, include at least two characters that are not letters, change passwords every one to six months, and choose character strings that do not appear in any dictionary. These guidelines are often supplemented with automatic checking of letter counts in typed passwords, with automatic expiration of passwords, and with benign background processes that attempt to crack passwords and notify users when they succeed. The entire authentication function can be moved to a separate server in the network, where the passwords and keys can be further protected and all systematic guessing can be monitored; examples of software that works this way include Kerberos, from the Massachusetts Institute of Technology, and Secure Remote Procedure Call, an option available with the operating systems of Sun Microsystems, Inc. Although they strengthen password security, the guidelines lead passwords that are hard to remember. So people ignore the guidelines, use easy passwords, and stick with their favorite passwords even after being asked to change. Spafford estimates that over half of all systems have at least one weak password (4).

Some system administrators recommend using "pass-phrases" as an easily-remembered way to generate strong passwords. A password generated this way consists of the first (or second) letters of a phrase or sentence. For example, the word string "quick brown fox jumped over the lazy dog" might yield "qbfjotld".

Spafford has proposed an approach called OPUS (for obvious password utility system) to eliminate weak passwords (4). OPUS functions as part of the program that allows users to change passwords. It checks whether the proposed password (and simple transformations of it such as reversal, initial capital, or all upper-case) are in an on-line dictionary. If so, the user is asked to propose a stronger password. Unfortunately, the database for this stratagem can be quite large and slow to search: a large dictionary of 250,000 words can occupy three or four megabytes of storage. Spafford proposes a compression scheme, also used in spelling checkers, that requires about 300 kilobytes of storage, making this a more practical scheme (Figure 2).

Although a scheme such as OPUS will strengthen systems against password attacks over the network, it may weaken systems to local attack. Strong passwords are hard to remember. Users write them down.

   //FIGURE 2 ABOUT HERE//

Password Replay

Since about 1990, replay has moved from a theoretical possibility to a practical reality; it now rivals password cracking as a means of entry. Attacks of this kind are familiar to users of cell phones and garage door openers, whose broadcast signals are accessible to anyone with a proper scanner. The same situation arises in the digital world among the computers connected by an Ethernet, which is a single, shared data-carrying cable. The sniffer is like the scanner: it sits in one of the computers and monitors all the traffic broadcast on the Ethernet. A common form of the attack is to watch for packets initiating a login and then record the next 128 bytes or so, sufficient to capture the user's name and password. The attacker comes by later to "harvest" user names and passwords from the sniffed data.

Many users of local networks feel helpless when they learn that it is easy for attackers to attach sniffers to their Ethernets. The best defense is end-to-end encryption, as in the Kerberos system, which requires significant redesign of operating system kernels. Very good defenses can be built from smart cards and retrofitted into existing systems.

One-Time Passwords

The source of these difficulties is the fundamental premise underlying password schemes since 1960: the reusability of passwords. A technology that allows one-time passwords would not be subject to attack by password guessing or replay. Suppose, for example, that a password is a six-digit number that changes randomly minute by minute. The attacker now has one chance in a million of guessing the number. The attacker cannot make headway by repeated guesses, since at best a few hundred guesses can be processed by typical login systems in a minute. An instant-replay attack is useless because the system will not allow a one-time password to be used a second time.

Leslie Lamport of Digital Equipment Corporation's System Research Center proposed a mechanism for one-time passwords that could be implemented in software. The mechanism was later released for Unix systems by Bellcore under the trade name "S-Key". The user generates a sequence of passwords, say P1, P2, ..., P100, where each one is obtained from its predecessor by a one-way cipher; for example, P5=E(P4). The final password P100 is given to the authentication server. For the next 100 logins, the user works *backward* through the sequence, and the authentication server remembers the last password used. The server allows login if the latest password, when enciphered by the one-way function, yields the last password used. A user of the S-Key system needs to install the S-Key software on his personal computer, which can then issue the proper next password in the sequence each time he logs in.

The technology of smart cards -- credit-card size computers -- is now mature and cheap enough to provide an even simpler mechanism for using one-time passwords. Since people are used to carrying cards and keys with them, a key granting access to one's computer is would be a minor addition that would greatly improve their ability to protect their valuable data. Smart cards that require PINs (personal identification numbers, a form of password) are also available for those who worry that a stolen smart card might be used to access their accounts. A thief trying PINs on a stolen card would take longer to break into the card than the system administrator would take to invalidate that card.

At login, the smart card's owner enters the PIN (if any) on the card's keypad, then follows the authentication protocol by typing what the card says when the host system asks for the card's response. (The smart card can also be embedded in a PC card that plugs into the PCM slot of a laptop computer, in which case the regular keyboard can be used for the PIN.) Because the card's responses are all signed by the card's private cryptographic key, the authentication server can always tell the name of the owner of a card to which it is speaking. It simply compares the number from the user's card with the number it expects, allowing login if there is a match (Figure 3).

   //FIGURE 3 ABOUT HERE//

One protocol for smart-card access was offered as a commercial product in the early 1980s by Sytek, Inc., of Mountain View, California. Sytek's protocol has come to be known as the challenge-response protocol because the user is asked to correctly respond to a number given by the computer. Sytek's product was popular among insurance companies to authenticate adjusters calling in from the field to request checks. Challenge-response cards are now offered by Racal-Guardata of Herndon, Virginia; Atalla, a division of Tandem Computers; and Digital Pathways of Mountain View, California.

Another type of card uses a time-series protocol. It is offered by Security Dynamics, Inc., of Cambridge, Massachusetts. The card displays the current time enciphered under the user's key. A resynchronization protocol is available to prevent failure that would result from drift between the card's and system's clocks. Similar protocols are offered by Racal-Guardata and by ThumbScan, Inc., of Lombard, Illinois. These cards have the advantage of being exceptionally low-cost because they have no keypad and require no smart-card reader.

Within a few years, desktop computers will include smart-card slots
and built-in smart-card authentication protocols. More advanced cards
will be activated by personal features of the holder instead of by PIN.
The activated card will execute the protocol automatically after it is
inserted into the slot. A strong push for adoption of such technology is
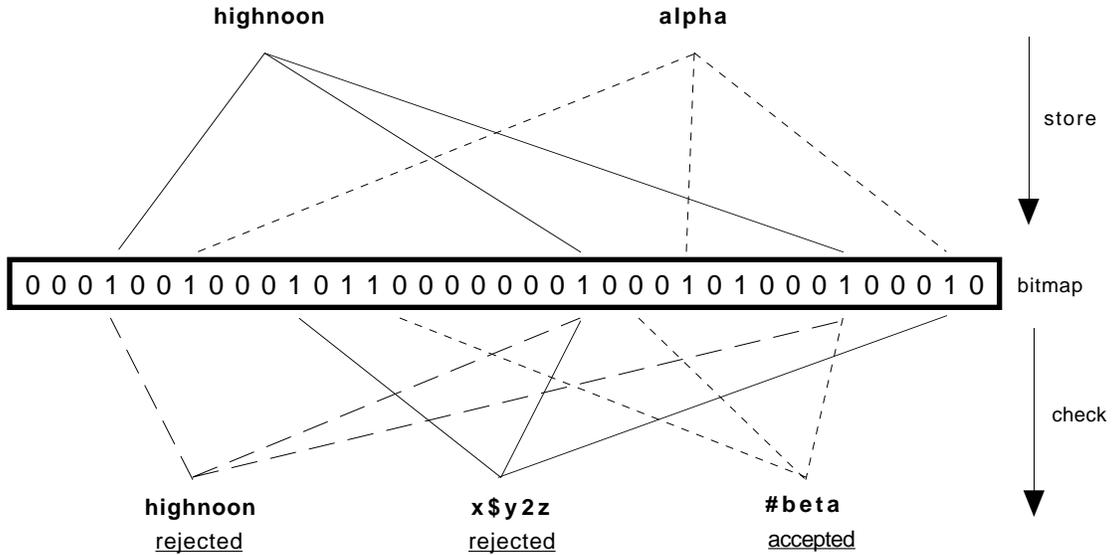likely to come from banks and credit card companies anxious to promote

commerce in the Internet. Their cards will provide other, related services, such as storing an address book, medical information, or even serving as car keys.

References

1. Robert Morris and Ken Thompson. 1979. "Password security: A case history." Communications of the ACM 22, No. 11 (November). 594-597.

2. Peter Denning. 1989. "The Internet Worm." American Scientist 77, No. 2 (March-April). 126-128.

3. Simson Garfinkel and Eugene Spafford. 1991. Practical UNIX Security. Sebastapol, California: O'Reilly & Associates.

4. Eugene Spafford. 1992. "OPUS: Preventing weak password choices." Computers & Security 11, No. 2. Elsevier.
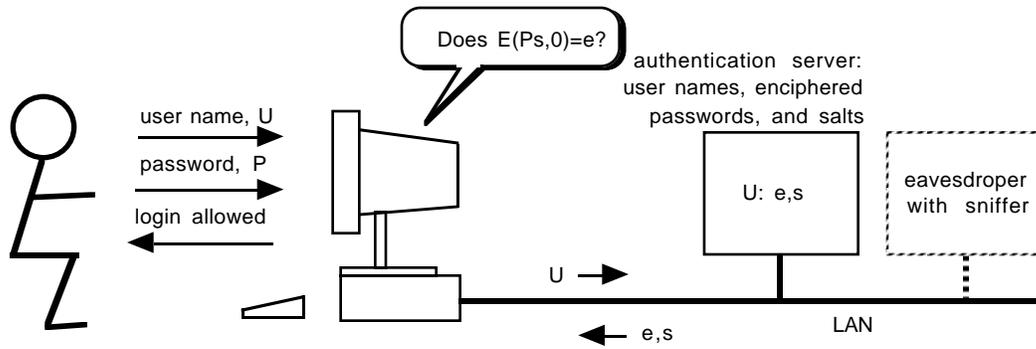
FIGURE 1. Access control in a local area network usually involves the storing and checking of enciphered passwords. A "salt" -- a stored bit pattern that can be appended to the password -- makes breaking into the network more difficult. At the request of the login program residing in the workstation, a user types a name and then a password. The password stays in the workstation; the login programs sends only the user name over the network to the authentication server, where the password file is stored; the file contains a list of authorized user names, enciphered passwords, and salts. The enciphered password and salt corresponding to the user name is sent back to the workstation; the workstation performs its own encryption, comparing the result with the enciphered password-salt combination arriving from the server. An n-bit salt increases the difficulty of password cracking by a factor of 2 raised to the n power: The attacker is forced to combine each guess with all salt values before enciphering and testing. An eavesdropper with sniffer can gain login later by replaying recorded bit-strings U, e, and s at the proper moments.

highnoon   **alpha**

store

0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0   bitmap

check

highnoon   **x $ y 2 z**   **# b e t a**
<u>rejected</u>   <u>rejected</u>   <u>accepted</u>

FIGURE 2. Compressed dictionary can be created for rapidly checking whether a proposed password is weak -- that is, whether it might easily be guessed by a password cracker. A series of fixed keys is used to assign familiar words a signature in the form of a set of positions in a bitmap; in this example the possible passwords "highnoon" and "alpha", along with others, have been assigned three bitmap locations. A proposed password is considered weak if its signature is in the bitmap. The proposal "x$y2z" has been rejected as weak even though it is not in the dictionary, because its signature happens to match a pattern in the bitmap. Such false positives cause little harm in this context. A password-changing program using this approach is designed to reject weak proposals.
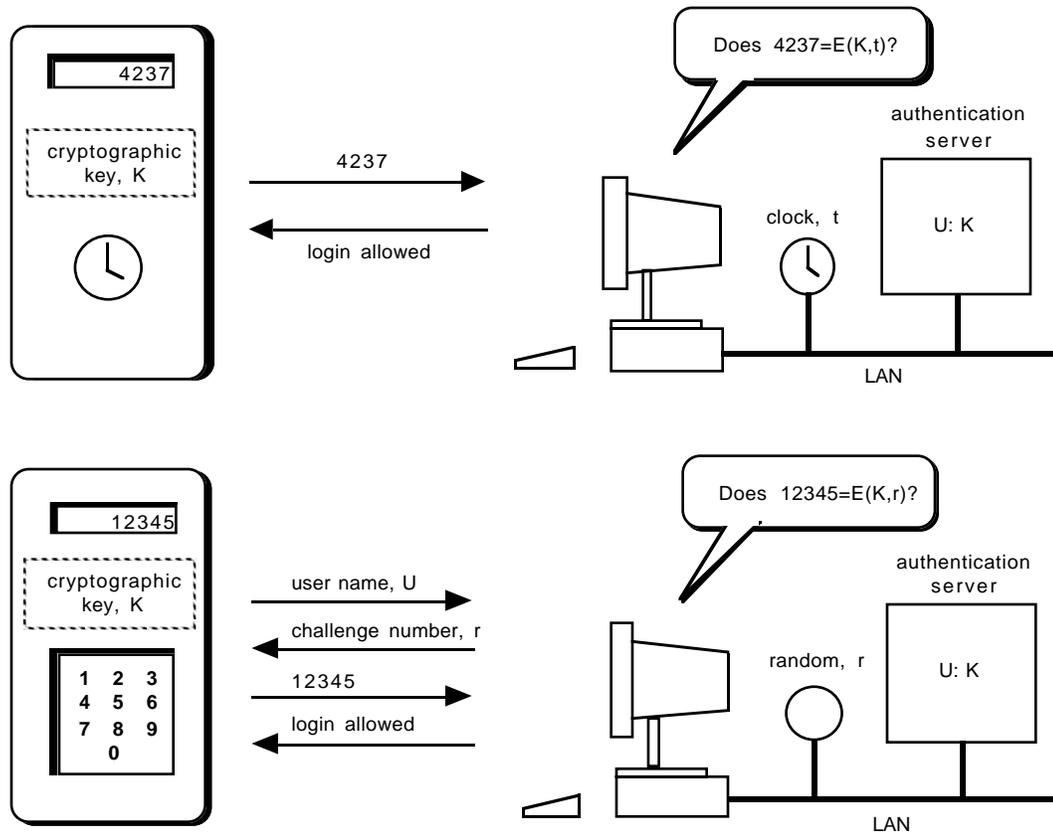
FIGURE 3. Smart cards provide a way to obviate password cracking and replay by generating one-time passwords that a user can type into a system. Two schemes are shown. The upper example shows a time-series protocol. The card is programmed to display a new number in a window every minute; the number is the time according to a clock in the card, enciphered under a cryptographic key assigned to the card. The card's owner transfers the number onto a system keyboard, and the system checks whether there is a user whose key produces that number when enciphered with the time from the system clock. The lower example shows the challenge-response protocol. The system gives the user a random challenge number. When this number is typed on the smart card's keypad, the card uses its cryptographic key to compute a response, which the user types to the system. The system checks that the response is the one that would be provided by the user's card when applied to the challenge number. The challenge-response protocol requires a card with a keypad, but it is immune to problems that would result from drift or failure in the system clock.