Lecture 9 : October 5, 1995

Lecturer: Ron Rivest *Scribe: Phil Lisiecki*

1 Outline

Throughout history, people have needed ways of identifying other people. Computers, similarly, need some method to authenticate users. There are a variety of techniques known and used, which involve techniques such as passwords, challenge-response, and biometrics.

2 User authentication

2.1 Purposes

There are several reasons why user authentication is needed.

Access control. Computers may be required to control access to computer logins, buildings, or restricted databases.

Authorization. Transactions such as electronic payments require the user to be authorized to prevent fraudulent transactions.

Auditing. It is often useful to log who did what, even if no controls are in place. Such logs facilitate accountability. If any malicious activity occurs, the pertretrator can be held responsible. This is fairly important, especially since most attacks on a computer system are from authorized insiders.

2.2 Principles

Authentication systems can be based on any of three basic things:

- Something the user *KNOWS*. This is a *password.*

- Something the user *POSSESSES.* A device serves as a *"token."*
- Something the user *IS* (or how he behaves). This principle is known as *biometrics*, or the measurement of some biological property of the user, such as hand shape or fingerprints. There is apparently a kindergarten at Lotus that uses hand shape to allow kids in and out.

For even greater assurance, two or more methods based on different priciples can be *combined.*

2.3 Considerations

Designing or choosing an effective user authentication system requires weighing many considerations.

Resistance to deceit, conterfeiting, circumvention. Reasonably determined attackers should not be able to fool the authentication system.

Time to authenticate. The user should not be required to spend too long interacting with the system. Passwords take only a couple seconds, while signing the user's name long-hand or using a retinal scanner take somewhat longer.

Cost of devices, interfaces, distribution, protection. These change a lot with technology. Currently passwords are very cheap, but DNA analyzers are not. This will likely change, someday, though.

Time and effort involved in updating. Keeping the user's authentication information up to date, or adding a new user's authentication information, can be time-consuming. This is especially true of systems like voice recognition which may require many training runs to be able to identify the user's voice.

Processing required. Often CPU time to process the information can be a problem. An authentication technique that took a minute to complete the computations would be disliked by users.

Reliability; maintainability. Many techniques suffer from reliability problems. For example, a finger print reader might get a smudge on it and become unable to read finger prints correctly, thereby denying an authorized used access.

Accuracy. Accuracy is usually measured as two quantities, the *false accept rate* and the *false reject rate.* As their names imply, the false accept rate measures the percentage of unauthorized users that get past the security, while false reject rate measures the percentage of authorized users that are denied their proper access. Note that this is often a trade-off: Making false accepts rarer makes false rejects more common, and vice-versa.

User acceptibility, ease of use, psychological factors. It is important for the users to feel comfortable with the authentication system. This includes factors such as difficulty, effort, and time required to authenticate. It also includes psychological factors, such as a general uneasiness to use their finger prints, since fingerprinting is generally associated with criminal activity. People are also wary to stick their fingers into a mysterious box placed before them. People might also be wary of shining light in their eyes to perform a retinal scan.

Transferability. Passwords are easily transferable, while retinal scans are not. The desired transferability properties depend largely on the system, as giving out passwords can be a security risk, though may also save administrative overhead to grant other users the appropriate priveleges. Transferability also suggests more potential for theft.

2.4 History

The human species depends on recognizable identity. What would society be like if we were all isomorphic sardines? In fact, some theories of evolution emphasize the complexity of social situation, especially involving deceit, as an explanation for the evolution of the human brain.

King's messengers used to carry signet rings to identify themselves.

Some evidence has been found that the Chinese used fingerprints 2000 years ago. (See *Computer Security Handbook*, page 38.) The Bertillon system, which dates back to 1870, was used in the United States for criminal identification. This system identifies a person by a composite of many measurements such as height, trunk height, arm span (both arms outstretched), forearm length, finger length, foot length, and head width. A famous case in the U.S. in 1903 in which two "Willie West"'s with identical Bertillon measurements were found. The day after that, fingerprints won as the dominant form of criminal recognition. (See *Computer Security Handbook*, page 38.)

Passwords have long been used in war-time scenarios.

2.5 Passwords

Passwords are cheap, easy, and reasonably accurate, and consequently, are in wide use. Recall Richard Stallman's objetions to passwords in previous lecture (see Levy's *Hackers*, pages 415-418); he suggested that all users use a zero-length password.

Passwords should have two basic properties:

- Hard to guess.
- Easy to remember.

These requirements conflict! Something which is easy to remember has small entropy, and is consequently easy to guess.

Morris and Thompson did a survey (CACM, Nov. 1979, 594-597) of 3289 passwords, and found 2831 (86%) vulnerable for one of the following reasons:

15	were a signle ASCII character
72	were a string of two ASCII characters
464	were a string of three ASCII characters
477	were a string of four alphanumerics
706	were a string of five letters, all the same case (upper or lower)
605	were a string of six letters, all lower case. Note that $26^6 = 309M$, which is not too large

Other vulnerabilities to passwords include:

Common words. English or other languages.

Common names. TV, musicians, significant other, pet, friends, family members, nicknames.

Easily obtained information. Birthdays, license number, phone numbers, social security number, make of car, number or street where user lives.

Keyboard patterns. Something like "qwerty".

Passwords on other systems.

Permutations of these. Especially backwards.

The Internet Worm (November 1988) tried the following attacks:

- No password
- User name
- User name appended to itself
- Nickname
- Last name
- Last name spelled backwards
- Personalized 432-word dictionary
- Dictionary attack

Some good ideas for creating passwords:

- Mix upper and lower case
- Use something unprounceable, such as "stowishy."
- Include non-alphanumeric characters
- Mix numbers and letters
- Perform a systematic substition, such as $o \rightarrow 0$ or $l \rightarrow 1$
- Pick letters from a longer passphrase or sentence
- Computer generation. This generates hard-to-remember combinations. Also, program might have a bug. For example, Professor Rivest's wife, Gail, discovered that the lock given to her by her gym had the same combination as many others in the gym by accidentally opening the wrong locker with the combination she was given.

In general, the risk should be lower than other forms of risk. (I.e., we don't need to over do this.)

Passwords must be stored somewhere. The user must remember the password, either by memorizing it or by writing it down, which some argue is a bad idea, though is it really that bad? In addition, the computer must remember the password so that it can be checked when the user presents it. There are several applicable techniques:

In the clear on file system. This is not a very good alternative, as a user that gains access to the file has all of the passwords.

On a dedicated authentication server. This is somewhat better, though a compromise of the authentication server will still reveal users' passwords.

Encrypted. This way, compromising the password file will not reveal users' passwords. But, we don't need to be able to decrypt, so...

Hashed. Use a one-way hash function, such as MD5. When the user presents the password, it is hashed and compared against the stored value. Knowledge of the hashed password is inadequate to authenticate oneself to the machine. Another useful trick that can be layered onto a hash is *salting.* Salting involves storing a value $hash(pwd, salt), salt$, where $salt$ is a per-user (possibly randomly chosen) value. Salting prevents pre-computation of hashes by an adversary, which makes breaking more common passwords at least a little more difficult. Furthermore, reuse of passwords, either by two users with the same password or one user with the same password on two systems, will not be evident from the salted hashes.

With access controls. It is often useful for all users to be able to access some of the information in the password file, but not have access to the actual passwords. Many systems break the password file into two pieces, one with useful user information, such as the user's default UNIX shell, and another "shadow" password file that is stored in a secret place that contains the actual passwords.

Changing passwords is another important issue. Changing passwords frequently improves security, though it makes passwords harder to remember. And, if very good, hard-to-remember, hard-to-guess passwords are used, there will be even more resistance to changing passwords. DoD says that passwords should be changed at least once a year. It is also generally not a good idea to reuse passwords, especially recent ones. Some systems do not allow a user to change their password to one recently used. Also, some systems restrict the frequency of password changes, while other require periodic changes. It is also generally advisable to change a password after using it in a way which might have compromised it. For example, Jeff Schiller, who works for MIT DCNS, always changes his password when he returns ¿from vacation, since he might have compromised his password by logging in remotely. As usual, a risk analysis should be performed to find the optimal compromise.

2.6 One-time passwords

The fundamental idea behing one-time passwords is to provide the user with a list of passwords, each of which works exactly once. Since the passwords are only used once, eavesdropping is much less of a problem. The user will compromise his password when he types it, but the password will no longer be valid, so this is not a problem.

S/Key system

S/Key provides the user with a list of passwords that are formed of six English words chosen from a pool of 2^{11} words. The system generates a list of passwords, and the user carries this list around.

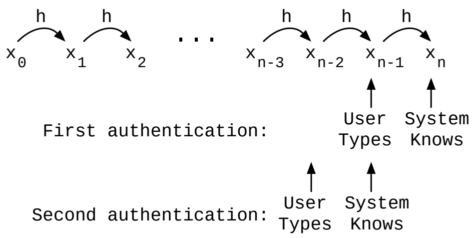

Figure 1: S/Key One-Time Password System.

The keys are generated by starting with a initial password, x_0, and then generating $x_{i+1} = h(x_i)$, where h is a one-way hash function. The system then remembers some password x_n. The user types the password x_{n-1}, which the system can verify by comparing $h(input)$ to x_n. If the comparison passes, the user is valid, and the computer saves the newly-typed password back into the password file. In this way, the user gives no information about the subsequent passwords, and not even the host knows what the next password will be.

One possible attack on S/Key is to masquerade as a host and ask for x_i, where i is much less than the value being used currently. The adversary will then know a large block of yet-to-be-used passwords, since $h(x_i) = x_{i+1}$, $h(h(x_i)) = x_{i+2}$, etc.

Of course, a man-in-the-middle attack will work here, but no password-based system can prevent that, since the connection that is established may be subsequently hijacked once the password exchange is complete, anyway.

SecurID

The SecurID system is an example of a time-based variable password. There is some possibility of masquerading, but this is only for a very limited time. The heart of this system is a small card that outputs a different sequence of numbers every 30 seconds. This number is computed as $y = f(time, secret)$, where $secret$ is a secret shared by the card and the host. The host can also compute y, and can then compare.

The essential properties of f are that it is a one-way function, and that relatively small changes in time create relatively large changes in the output. A hash function should be adequate for this purpose.

One major drawback to this system is that the card can break. Also, time synchronization can be a problem, though careful engineering should minimize that.

Another possible problem with the SecurID card is loss or theft. An adversary in possession of the card can authenticate himself as the card's owner. One possible solution is to require the user to enter a code into a keypad on the card. This would be tedious and would make the card more expensive and more easily breakable. Another possibility is to type a pin in after y, though this number could be seen by an eavesdropper on the network. A simpler, yet effective, strategy is to have the user add a pin to the random number digit-by-digit. The addition is not very difficult for the user, and because y is not known to the adversary, no information about the pin is disclosed to the adversary.

Using SecurID effectively requires that only one login attempt be allowed per 30-second interval, since an eavesdropper could reuse a password to start a second session if he were quick.

2.7 Challenge-response system

Challenge-response systems ask the user a question or sequence of questions which must be answered correctly.

One way of implementing this is to use a token, like Bell Labs uses. Basically, the host shows a challenge x, the user types the challenge into his device, the device computes $y = f(x, secret)$, and the user types the response back to the host. This sort of system removes the time synchronization problems inherent in SecurID and eliminates the need to carry around lists of passwords. Also, multiple, concurrent logins can occur without risk of an adversary using the authentication information from one login to falsify a second login. The user is still, however, at the mercy of a token that can break, and this system has the additional disadvantage that there is much more work for the user to do.

There was some discussion of whether counting is a good way to find the value of x. One advantage is that if x jumps, the user knows that someone has been trying to access his account. The device could be smart enough to check for out of sequence queries. On the other hand, if a masquerader could convince a user to give a higher-numbered password, then the adversary would have access to the account at some point well into the future. If, instead, x were random, a masquerader would have no idea what challenge to present to obtain any useful information.

Another type of challenge-response system is to ask the user a sequence of personal questions, such as grandmother's maiden name, name of first pet, best friends' favorite food, etc. This is reasonably convenient for the user, though most of this information can be researched, and such answers have fairly low entropy.

3 Handouts and Readings

Problem set 4 was handed out. The first problem on it is an attempt to mimic Professor Rivest and Rosario's exploration of the BAN logic system presented in the last lecture. A reading packet containing *A Performance Evaluation of Biometric Identification Devices* by Holmes et al and *The S/KEY One-Time Password System* by Neil Haller were handed out.