CHAPTER 1

The Authentication Landscape

Open, Sesame!

— Scheherazade (attr.), *Ali Baba and the Forty Thieves*

IN THIS CHAPTER

This chapter provides an overview of *authentication*, the problem of verifying identities, and the major issues in making it work.

- Elements of authentication systems
- Early developments in password authentication
- Attacks via cleverness, theft, and trickery
- Authentication factors: passwords, tokens, biometrics
- Judging attack prevalence
- Summary of the chapter's attacks and defenses

1.1 A Very Old Story

For centuries, people have relied on guards, spoken passwords, and hard-to-forge seals to prove their identity to other people and to verify important messages. Unattended authentication by mechanical devices is also quite old: key-operated locks date back to the ancient Egyptians. Practical mechanisms did their job with as little human interaction as possible.

Unattended authentication is essential with today's computer-based systems. It may be a cliché to call the Internet the "information superhighway," but here it captures truth: we can't afford to post a policeman at every cloverleaf or on the countless interconnected streets and driveways. We must depend on mechanized protection.

The notion of an unattended, password-controlled lock appeared centuries ago in *Ali Baba and the Forty Thieves*, the Mideastern folk tale. In the well-known story, the narrator Scheherazade told of a great treasure hidden in a cave behind a stone. The password "Open, Sesame" caused the stone to move out of the way. Guards in cities of that era also used passwords to allow citizens through the city gates. But the thieves' cave didn't need a human guard to recognize faces, voices, or styles of dress. Instead, there was an unexplained and probably magical device that mechanically responded to the spoken words. Most importantly, the mechanism didn't discriminate between different people speaking the words. It responded to the words themselves, just like modern combination locks or password-protected workstations, which admit anyone knowing the secret. ***see Note 1.***

The point of Scheherazade's tale is that magic (or mechanism) always follows its own logic, independent of people's wishes or intentions. The same thing plagues us today with computer-based authentication systems. We have a wealth of technical alternatives, each following its own logic and falling to its own distinctive weaknesses. But if we understand the logic and the weaknesses of a given method, we stand a better chance of bending technology to fulfill our real needs.

The passwords and other authentication mechanisms used with computers today cover a broad range of techniques and technologies. Web site designers, e-commerce planners, and other system developers must choose from numerous products and make numerous configuration decisions within each product. Systems like Windows NT and Windows 2000 by themselves incorporate several password alternatives to provide interoperability with other products. Some organizations need the extra security of smart cards or authentication tokens like Safeword or SecurID. A major motivation behind the "public key infrastructure" (or PKI) is to someday revolutionize, strengthen, and simplify individual authentication. But, as with any evolving technology, it's hard to predict how much of its promise it will ultimately achieve.

TABLE 1.1: *Examples of the Five Elements in an Authentication System*

Authentication Element	Cave of the 40 Thieves	Password Login	Teller Machine	Web Server to Client
Person, principal, entity	Anyone who knew the password	Authorized user	Owner of a bank account	Web site owner
Distinguishing characteristic, token, authenticator	The password "Open, Sesame"	Secret password	ATM card and PIN	Public key within a certificate
Proprietor, system owner, administrator	The forty thieves	Enterprise owning the system	Bank	Certificate authority
Authentication mechanism	Magical device that responds to the words	Password validation software	Card validation software	Certificate validation software
Access control mechanism	Mechanism to roll the stone from in front of the cave	Login process, access controls	Allows banking transactions	Browser marks the page "secure"

1.2 Elements of an Authentication System

Regardless of whether an authentication system is computer based or not, there are several elements usually present, and certain things usually take place. First of all, we have a particular *person* or group of people to be authenticated. Next, we need a *distinguishing characteristic* that differentiates that particular person or group from others. Third, there is a *proprietor* who is responsible for the system being used and relies on mechanized authentication to distinguish authorized users from other people. Fourth, we need an *authentication mechanism* to verify the presence of the distinguishing characteristic. Fifth, we grant some privilege when the authentication succeeds by using an *access control mechanism*, and the same mechanism denies the privilege if authentication fails. Table 1.1 gives examples of these elements.

For example, the person of interest to the thieves' cave might be Ali Baba, his brother, or whichever thief wanted the door to open. The distinguishing characteristic was knowledge of the password, "Open, Sesame." The cave's proprietors were obviously the gang of the Forty Thieves. There was some unexplained authentication

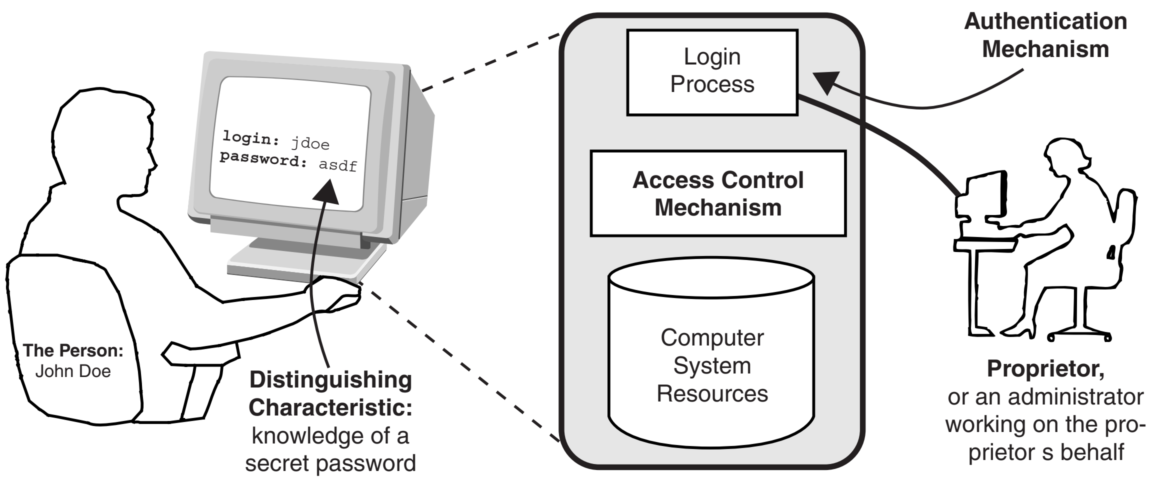

login: jdoe
password: asdf
The Person: John Doe
Distinguishing Characteristic: knowledge of a secret password
Login Process
Access Control Mechanism
Computer System Resources
Authentication Mechanism
Proprietor, or an administrator working on the proprietor s behalf

FIGURE 1.1: *Five elements of authentication.* These elements are: the person, the distinguishing characteristic, the proprietor, the authentication mechanism, and the access control mechanism.

device in the cave to identify the correct password and to ignore incorrect ones or, presumably, general conversation (the story did not say what would happen if a thief mentioned the password in a conversation near the cave door; probably the stone would have rolled out of the way). The access control mechanism moved the stone which granted access to the cave.

A genuine example is, of course, the password-controlled login operation we encounter in most computing environments. The person of interest is an individual allowed to use the computer. The system usually assigns the person a symbolic name or user identification code which we will call the *user name*. For example, John Doe is an authorized user of the system in Figure 1.1, and the proprietor has assigned him the user name “jdoe.” The distinguishing characteristic for John Doe is his secret password, “asdf.” The process should be familiar: John gets the computer’s attention and the computer’s login process prompts him for a user name and a password. The process contains an authentication procedure that compares the typed-in password against the password established by or for John Doe; the procedure succeeds if the two match. The access control mechanism allows John to proceed with using the system, and the system uses John’s user name whenever it makes access control decisions on protected resources.

When looking at computer security problems, we always need to distinguish what we want to do from what we really do. The former question, “what we want,” is usually spoken of as *security objectives*. The gang of the Forty Thieves, for example, had the objective of protecting their loot from theft. They relied on a *security mechanism*, the cave’s door, to do this. In a computing system, the proprietor has the objective of granting access only to authorized users. In Figure 1.1, the proprietor relies on the operating system, and its password controlled login, to achieve this objective.

As a practical matter, there’s always a gap between what we want and what really happens. A lock lets anyone in, as long as they have a copy of the right key. The lock does not keep the wrong people out unless we can prevent the wrong people from having a key. That can be hard to do, especially if the people we lock out really want to get past that door. Moreover, we can’t always afford to put separate locks on everything. Often there’s just a big lock on the outer door, and we have to trust the people we’ve allowed inside.

Computer systems usually provide authentication and access control as clearly separate things. While it sometimes makes sense in the mechanical world to distinguish between the bolt that holds the door shut and the lock that controls the bolt, such things are often built into a single mechanism. On computers, the authentication process establishes the correct user name to use, and access control happens separately. Computer systems generally control access by comparing the person’s user name with the access rules tied to a particular file or other resource. If the rules grant access by the person with that user name, then the person gets to use the resource.

The Forty Thieves intended their cave to grant access only to members of the band, but the mechanism couldn’t prevent others from using the password. This problem infects both authentication and access control. In authentication, we can identify the people we want to allow to use a system, but the mechanisms aren’t perfect. There’s always a way for an unauthorized person to masquerade as a legitimate user.

We have a similar problem in access control: we want to authorize certain people to use the system, and we implement those desires by setting up the access control system to allow this. In an ideal security engineering world, we grant access using the principle of "least privilege," in which people have just as many permissions and privileges as they need: no more, no less. But in the real world, the access control system can't give people exactly the privileges they need: we must either give them too many or omit a few that they really need. In a practical world we usually extend a measure of trust to authorized users so that they have the tools to get their work done, even though this technically gives them permission to do things they shouldn't be doing.

Access control can be very complex, even without trying to achieve least privilege. Modern computing systems provide a broad range of access control policies and mechanisms. Even the access control mechanisms provided by relatively common systems like Unix, Windows NT, or Windows 2000 allow users and administrators to establish very complicated sets of rules for granting or denying the use of various computer resources. However, many organizations take the relatively simple approach of tying access control and authentication together, so that authenticated users have only a few broad access restrictions.

Although the problem of authenticating people poses a real challenge to computer systems, they aren't the only entities we need to authenticate. We also need to authenticate unattended computer systems like Web servers, especially when we ask them to perform an expensive service. Unlike user authentication, there isn't really a person standing at the server for us to authenticate. Instead, we want to ensure that we speak to the right piece of equipment under the control of the right people or enterprise. We don't want to order boots from a computer claiming to be "L. L. Bean" unless we will receive the boots. When we authenticate L. L. Bean's server, we need confidence that its distinguishing characteristic is managed and controlled by the L. L. Bean enterprise. Usually, the browser warns its operator if it can't authenticate the site, and leaves the access control decision to the operator ("Should I still order boots, even though this site doesn't really seem to be L. L. Bean? I don't think so!"). In a sense, the process turns the automatic authentication function upside-down, but the underlying concepts are still the same.

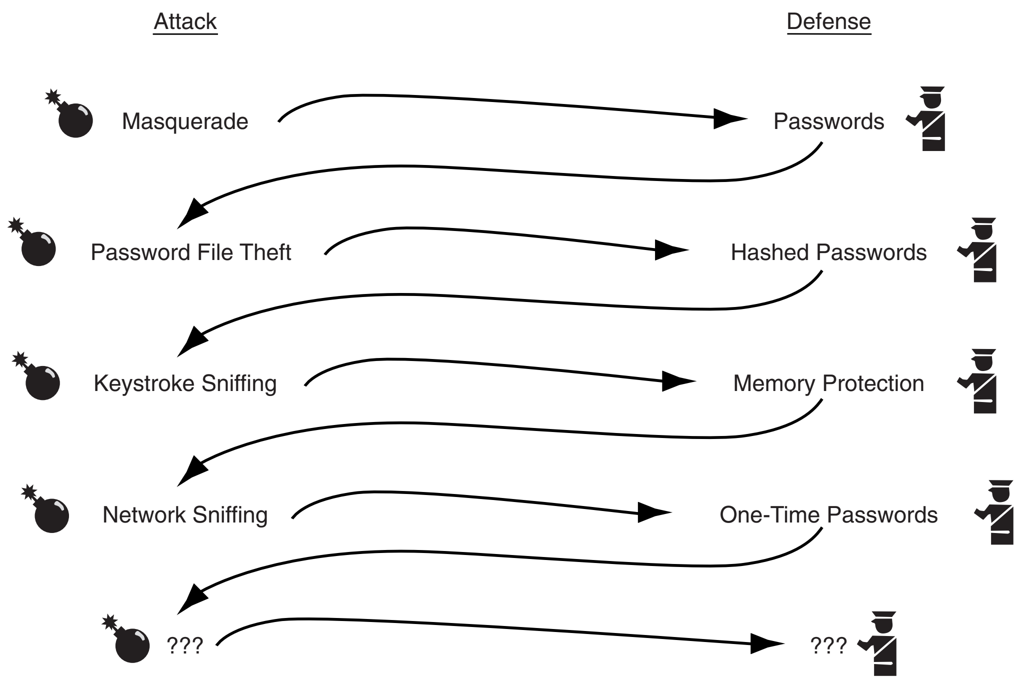


FIGURE 1.2: *Attacks and defenses evolve in response to each other.* As attacks develop, defenses develop in response. Newer attacks evolve to circumvent the new defenses. The examples shown here tell only the beginning of the story.

REVISED ATTACKS AND REVISED DEFENSES

Today's authentication systems evolved from decades of attacks, many of them successful. Starting with password systems in the early days of timesharing, authentication systems have been under constant attack. The earliest attacks were from members of the local user community protesting against the notion of authentication and access control. Early success at stealing password files led to a defensive measure called *password hashing*. This led to attempts to intercept ("sniff") passwords, which in turn led to other defensive measures. Figure 1.2 illustrates the general progression of increasingly sophisticated defenses in response to increasingly sophisticated attacks. While the figure doesn't represent an accurate historical time line, it accurately captures the dynamics of an ongoing competition between proprietors and attackers. Later sections of the book describe these attacks and defenses in detail.

Several of these attacks appear in the next section on the evolution of passwords in timesharing systems. Each attack description is marked with a numbered "bomb" icon in the right margin, shown here. The attack's number begins with "A-" and is keyed to the summary of attacks at the end of the chapter. The "uniformed guard" icon in the right margin indicates the description of a defense, usually against a recently described attack. Again, the defense's number is keyed to a summary of defenses at the end of the chapter. As suggested by the figure, the bombs and guards tend to alternate throughout the book.

A-n

D-n

SECURITY STRATEGIES

An important question to ask about any defensive measure is whether or not it is truly necessary in a given situation. Although this question often hinges on technical questions (i.e., does the defense do its job in the particular situation?), it also hinges on questions of organizational policy and the motivation behind its security activities. There are three general policy rationales to justify security measures:

- **Standards of due care**

  This is a legalistic concept. Businesses are legally obligated to install well-known safety measures to protect against well-known risks. Court cases have found businesses negligent for failing to do so in other industries, although there are as yet no such precedents involving information security. However, some information security measures are so common and so well known that they are obvious candidates for representing standards of due care. Anything noted in this book as an *essential defense* should be considered a minimum standard for due care and should always be present. ***see Note 2.***

- **Risk analysis**

  This approach is based on a cost/benefit trade-off. A business determines that it should install security measures (at a particular cost) by estimating the losses it might incur from likely attacks. This computation is often called *risk analysis*, and the U.S. government published a standard describing the process (the standard was withdrawn in 1995). Because of management and budgetary inertia, enterprises rarely perform such assessments until after a significant loss occurs. Occasionally, an enterprise learns from the mistakes of its peers and installs a defense before they themselves are attacked. *see Note 3.*

- **Exceed industry practices**

  In this approach, a business tries to avoid attack by posing a slightly more challenging target than its neighbors. For example, most banks use conventional passwords to protect all on-line transactions, even for large corporate customers. A bank can deter certain types of fraud from large accounts by adopting one-time passwords. In theory, this should encourage attackers to turn their attention to other banks, since weaker security measures are easier to overcome.

  In some regions this is called the "bear chase" strategy, based on an old adage: when a bear chases your group of hunters, you yourself don't have to outrun the bear, you only have to outrun the slowest hunter. In practice, some companies do this after a peer suffers a serious loss: they install a defense against the same type of attack before they suffer loss themselves.

The best balance for most enterprises probably combines these approaches. By meeting standards of due care the enterprise can deter claims of negligence. By implementing industry practices the enterprise avoids drawing attacks by being perceived as an easy target. By exceeding expectations, the enterprise poses an unpredictable target for potential attackers. Even if the enterprise can't afford to do a full cost/benefit risk analysis, there are often a few obviously risky areas where improved security pays for itself.

Today, the right authentication choices for a particular enterprise or application depend on how people use the systems in question, how the systems are built, and what types of attacks they expect. We explore those choices by looking at how well different authentication systems have worked over the years and what problems persist today. The authentication capabilities of today's commercial systems, and the promise of tomorrow's evolving systems, all stand upon our past successes and failures.

1.3 Authentication in Timesharing Systems

We start with a look at timesharing systems because they have a lot in common with modern server systems and because they hold the genesis of modern password systems. Just as cheap locks remain popular for desks and cabinets, passwords will always play some role in computer-based authentication. Today, they are wildly popular on Internet Web and e-commerce sites. While password security isn't foolproof, modern systems reflect many lessons learned from the days of timesharing systems.

In the earliest days of computers, the computer itself didn't have to handle access control and authentication. People either worked with the computer directly (if they could unlock the computer room door) or they submitted computer programs to other people (computer operators) to run the programs on their behalf. This changed in the 1960s with the advent of timesharing systems, which were the first interactive "server" systems that provided services to lots of different, noninterchangeable people simultaneously.

The Compatible Time Sharing System (CTSS) at the Massachusetts Institute of Technology (MIT) was arguably the first successful timesharing system. Its designers, notably Fernando J. Corbató, envisioned a system that handled a large and varied community with hundreds of users. Under such circumstances, Corbató saw the need for some degree of privacy and separation between different people's work. Moreover, computers back then were astronomically expensive by today's standards: a single second of borrowed CPU time could cost $100, and often cost more.

To provide the modest level of security Corbató thought sufficient for an academic environment, he proposed what seemed to him an obvious solution. Students generally stored personal items in metal lockers secured with combination locks: it was a simple matter to provide a similar, memorized "lock" for timesharing users. In 1963, the feature was added to CTSS. From then on, people had to type a memorized "private code" in addition to the user name that told CTSS how to find their personal files. ***see Note 4.***

Today, of course, we refer to Corbató's private code as a *password*. Figure 1.3 illustrates the basic mechanism. The computer asks the person to type in a user name and a password. The com-

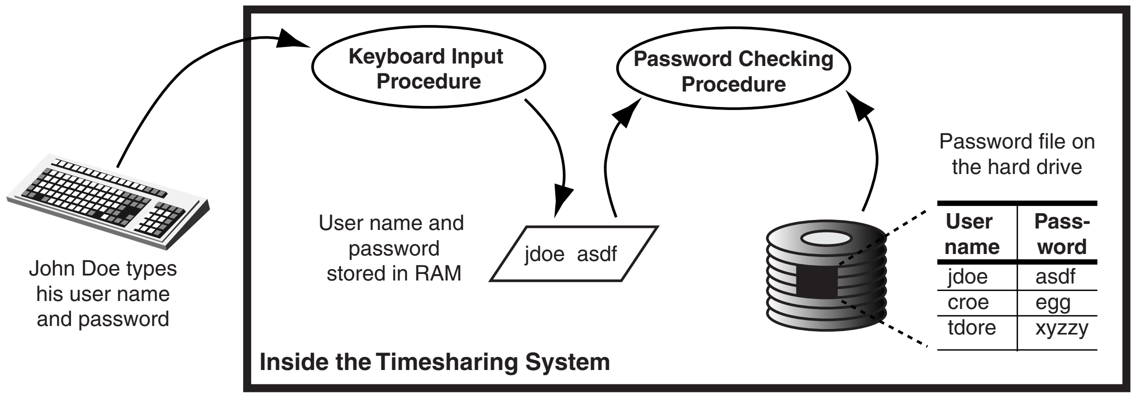

Keyboard Input Procedure
Password Checking Procedure
Password file on the hard drive
User name and password stored in RAM
jdoe asdf
John Doe types his user name and password
User name | Password
jdoe | asdf
croe | egg
tdore | xyzzy
Inside the Timesharing System

FIGURE 1.3: *Basic password checking on a timesharing system.* John Doe's user name and password are read into main memory. The user name is used to look up the password in the system's password file. The typed-in password must match the copy stored in the system password file, otherwise the procedure fails.

puter searches the system's password file for an entry matching the user name. If the password in that entry matches the password just typed, then the login succeeds.

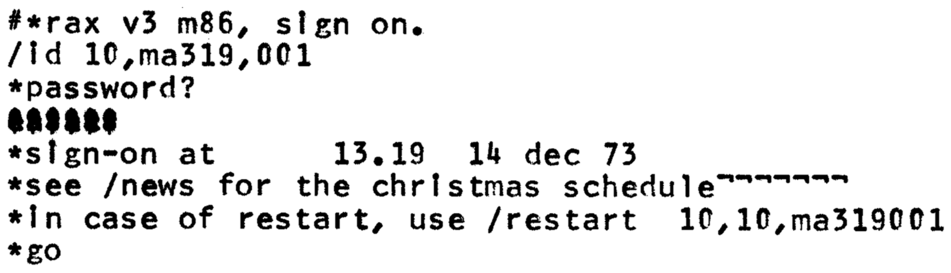
Some early computer users, particularly among the technically sophisticated ones called "hackers," did not like the notion of user names and passwords. They might have tolerated the restrictions of locked doors or computer operators, but the password mechanism was different. It took power away from them and put the computer itself in charge. This role reversal unsettled some and outraged others.

see Note 5.

PASSWORDS UNDER ATTACK

A battle of wits ensued. Programmers who missed the absolute control they had over the computer would probe the timesharing software, and the password mechanism in particular, looking for flaws that would give them back their lost power. The programmers responsible for the timesharing system would examine their own work to try to stay one step ahead.

Sometimes the timesharing programmers did stay ahead. Across the Charles River from CTSS, at Boston University's Remote Access Computing System (RAX), a timesharing programmer was looking



```
#*rax v3 m86, sign on.
/id 10,ma319,001
*password?
[illegible]
*sign-on at      13.19  14 dec 73
*see /news for the christmas schedule¬¬¬¬¬¬¬
*in case of restart, use /restart  10,10,ma319001
*go
```

FIGURE 1.4: *Logging on a timesharing system.* The timesharing system printed the first line to identify itself. The user typed the line starting with a slash to identify himself. The system asked for the password and overprinted several characters in a row so that other users could not read the typed password. This is called *password blinding*. The remaining lines were typed by the RAX timesharing system after the login succeeded.

over the program that checked a typed password (Figure 1.4). The program was unusually complicated because programmers had revised it several times to allow backspacing and other line editing functions to correct password typing errors. Suddenly the programmer realized that a peculiar sequence of backspace, tab, and line delete characters would cause the program to log the person on without checking the password at all. He managed to correct the problem before anyone else found out about the problem and tried to use it. ***A-1***

Other times, the hackers got the upper hand. Back at CTSS, members of the local hacking community found that the weak point in the password system was often the password file itself. The CTSS programming staff tried to build the system so that users could not retrieve the password file, since the file listed the names and passwords of all CTSS users. According to legend, a hacker would occasionally manage to extract a copy of the password file from its secluded location, print it out, and post the file on a nearby bulletin board for all to see. ***A-2***

The most notorious occasion, however, was blamed on a flaw in CTSS itself. One afternoon, an administrator was editing the password file at the same time another administrator was editing the daily message, which was automatically displayed whenever a user logged in. Inside CTSS, the editor program confused the temporary file containing the daily message with the temporary file containing the passwords. Whenever someone logged in, the system automatically displayed the password file to them. Naturally, the problem emerged late on a Friday afternoon and went unnoticed until after the administrators left for the weekend. It persisted until a thoughtful user invoked a hardware fault that crashed the computer. *see Note 6.*

In these early days, many timesharing programmers treated bug fixing as their principal defense against such attacks. While some considered passwords a practical but limited security technique, others really believed they could provide foolproof authentication, especially given the physical arrangement of typical timesharing systems (Figure 1.5). Any weaknesses were caused by fixable software flaws, not by fundamental weaknesses in the technique itself. This attitude hinged on two assumptions: first, the programmers believed they could identify and eliminate most, if not all, of the security flaws in the system; and second, they believed the system could reliably prevent users from reading the password file.

Security experts today would dispute both assumptions. By the 1960s, computing systems had become too complex to ever be bug-

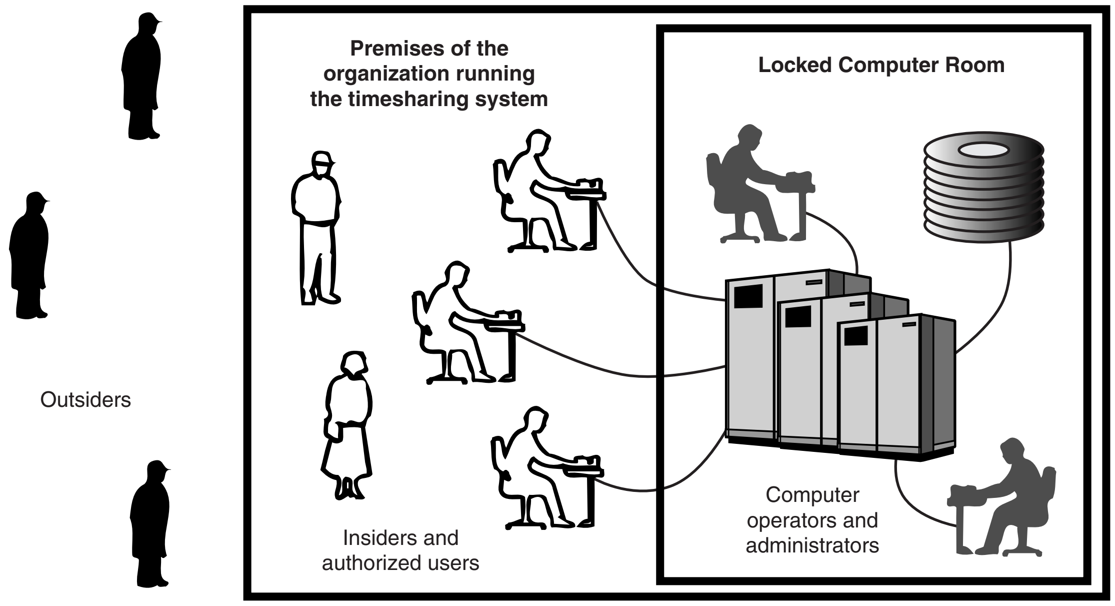

Premises of the organization running the timesharing system
Locked Computer Room
Outsiders
Insiders and authorized users
Computer operators and administrators

FIGURE 1.5: *Physical arrangement of a timesharing system.* All equipment connected to the earliest timesharing systems resided within the physical premises of the organization that ran the system. The computer itself was kept inside a locked computer room, following the older tradition of batch processing sites.

free. Large-scale software design was in its infancy; the RAX line editing problem might not happen today because the particular problem could be "designed out" of the password handling procedures. The same is true for the problem underlying the switched files on CTSS. ***D-1***

But these problems were replaced by other, unexpected ones. In practice it has proven almost impossible to unconditionally protect a file from unauthorized reading. In a classic attack, a user privileged to see a protected file, like the password file, could be tricked into running a program that secretly copied the file to an easy-to-reach location. That style of attack was named the *Trojan horse*. If any program on the system could read a secret file, then attackers could usually find a way for other programs to do the same thing. ***A-3***

HASHED PASSWORDS

In 1967, Cambridge University started running their Titan timesharing system on a continuous basis in order to provide a reliable, full-time computing service to the university community. Even in 1967, people knew it was essential to make back-up copies of files to protect against disasters. But these back-up tapes posed a security dilemma, since they held copies of Titan's password file. ***see Note 7.***

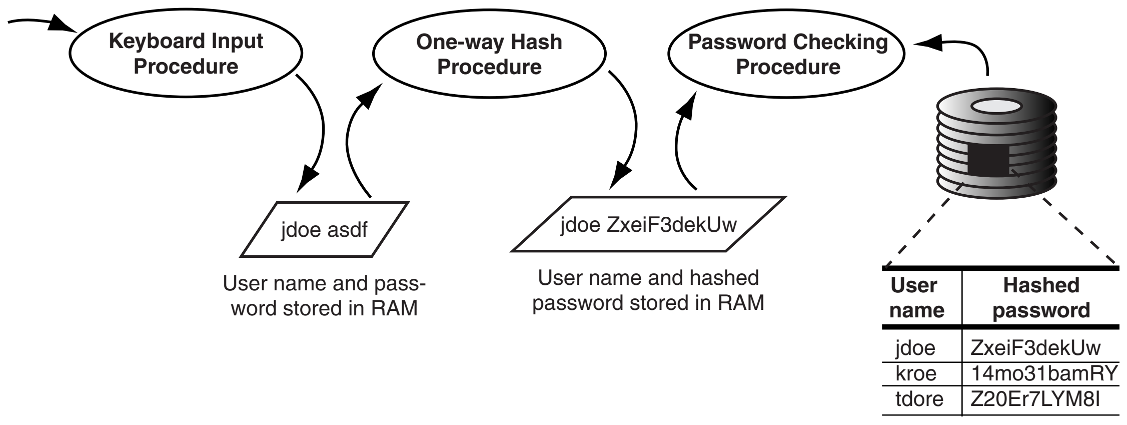
One evening, Roger Needham was sharing a few pints with another Titan developer, Mike Guy, and discussing the vulnerability of password files stored on back-up tapes. They struck on the notion of encrypting the passwords using a "one-way cipher" that would disguise the passwords in an irreversible way. The procedure converted a text password into an undecipherable series of bits that attackers couldn't easily convert back into the password's text. Mike coded up the one-way function and they installed the revised logon mechanism. ***D-2*** ***see Note 8.***

One-way functions are depressingly common in real life: it takes moments to injure or break something, but it can take hours or weeks to make it whole again, or stacks of money to replace it with a new copy. Automobile accidents provide an extreme case: it takes a lot of time, money, and material to build a new car, but it takes only a moment to "total" that car in a crash. Mathematics provides similar functions: we can easily combine numbers in various ways, but it might be difficult or impossible to figure out what numbers we started with.

The function that Guy and Needham installed in Titan is today called a *one-way hash*. Figure 1.6 shows how we use it with passwords. When John Doe logs on, he types his user name and password as usual, and they're read into RAM. Next, the system applies the one-way hash procedure to his password. Then the system extract's John's entry from the password file on the system's hard drive. The password entry contains a copy of John's password as it appears after applying the one-way hash procedure. If John typed the right password, the hashed copy should match the copy in the password file.

The one-way hash thwarts the objective of stealing the password file from the back-up tape (or from anywhere else), since attackers can't retrieve users' passwords simply by looking at the stolen password file. The technique is still used today: every modern server system stores passwords in hashed form. Password hashing has become an essential defense in any system that uses passwords.

A good one-way hash function has two properties. First, the function must compute a result (the *hash*) that depends on all of the input data. The hash result should be different, and often wildly different, whenever we make minor changes to its input. Second, there


Keyboard Input Procedure
One-way Hash Procedure
Password Checking Procedure
jdoe asdf
User name and password stored in RAM
jdoe ZxeiF3dekUw
User name and hashed password stored in RAM

User name	Hashed password
jdoe	ZxeiF3dekUw
kroe	14mo31bamRY
tdore	Z20Er7LYM8I

FIGURE 1.6: *Checking a hashed password.* John Doe's user name and password are read into main memory. A one-way hash procedure transforms the password into an undecipherable mass of bits. The password checking procedure compares the hashed version of the typed-in password against the hashed password stored with John's user name in the password file.

must be no practical way to convert the hash result back into the original data. Since the function accepts an arbitrarily large amount of input data and yields a fixed amount of output, it's going to be possible to generate the same output to two different hash results. Even so, there should be no easy way to find the input data that yields a particular hash result.

In fact, there should be no way to produce a particular result from a one-way hash function except by trying all possible input values until the desired hash result appears. Encryption procedures have a similar property: there should be no simple way to deduce the original plaintext data from the encrypted data unless one knows what encryption key was used (we will take a closer look at encryption in Section 5.3). However, encryption procedures are designed to be reversible, and that's why they use a key. We don't need to reverse the password hashing and, in fact, the passwords are safer that way.

A system that uses hashed passwords must perform the hash whenever someone enters a password into the system. First, the system hashes the initial password assigned to the user before storing it in the password file. Then every procedure on the system that collects a password from the user must hash it before doing anything further. The login procedure hashes the password before comparing it against the password file. The password changing procedure hashes the old password to authenticate the user before allowing the password to change, and then hashes the new password before storing it in the password file.

Many timesharing designers adopted password hashing over the next several years. Developers at MIT used the technique in the ambitious follow-on system to CTSS known as Multics, the "Multiplexed Information and Computing Service." However, the Multics experience illustrated the importance of using a good hash function. In the early 1970s, the U.S. Air Force began "Project ZARF," which probed the security of the early Multics system. The Air Force team quickly penetrated Multics file protection and stole a copy of the password file. After examining the hash function it used, they found a way to invert it. At a subsequent meeting with the Multics developers, the ZARF team presented a slip of paper to the author of the password software, and he found his password written on it. The security problems were largely cleared up in subsequent versions of Multics and, in time, it developed one of the best reputations in the community for operating system security. ***see Note 9.***

1.4 Attacking the Secret

Clearly, passwords will not work if people can steal them directly from the authentication system. But this isn't the only way of retrieving passwords. Attackers can also exploit the fact that most people do a bad job of creating and keeping secrets. This opens the door for guessing attacks and social engineering.

GUESSING ATTACKS

Although technically savvy attackers might go after a password file, other attackers might take a simpler approach: they exploit human nature to try to guess what passwords people use. In a typical case, the attacker develops a list of possible passwords and makes successive attempts to log on using the different passwords. We refer to this general strategy as a *trial-and-error* attack. Uninspired attackers might try every legal password in hopes of hitting the victim's password soon. Cleverer attackers construct a list of likely passwords, like the victim's name, the victim's spouse's name, the victim's kids' names, and so on. ***A-4***

Guessing attacks can succeed if people are careless about password selection and if trial-and-error attacks proceed without detection. The first defense against such attacks is to keep an *audit trail* of attempts to log on to the system. An audit trail is a record of significant events within the system and is a common feature in modern computing systems. Auditing in Unix is provided by the syslog facility, and both Windows NT and Windows 2000 provide similar features. All computing systems that meet U.S. government security requirements (traditionally defined by the *Trusted Computer System Evaluation Criteria*, or TCSEC) must provide an auditing mechanism. Typically, an audit system will record the date, time, user name, and specific details associated with each audited event. ***D-3*** ***see Note 10.***

Unfortunately, detailed audit records can actually cause problems with password mechanisms. In most cases, a good audit mechanism will record enough information so that someone reviewing the audit later can reconstruct what happened in great detail, even to the point of understanding the types of mistakes users have made. Clearly, however, audit records about password authentication should never include the typed password, whether the password was correct or not. If a potential attacker reviewed the audit log, misspelled passwords would clearly provide strong hints as to the actual password.

Even basic audit records can unintentionally leak passwords when they record the user name. Occasionally, when people respond to a prompt to log on, they enter the password when they should enter the user name. For example, imagine what would happen in Figure 1.1 if John mistyped his user name and/or password a couple of times in a row. If he types the return or enter key too many times, he may find himself typing his user name into the password prompt and his password into a subsequent user name prompt. If the system generates an audit record of this mistake, the record will contain his password. Then an attacker can retrieve John's password by reviewing the audit log. ***A-5***

The typical defense against this is to treat password auditing as a special case. While the audit software normally attempts to record the user name associated with an event, we must admit that we really don't know what user name to associate with logging on until the operation has succeeded. We achieve better security if we focus on trying to detect guessing attacks instead of burying the password attempts in audit records. A typical strategy, used by systems ranging from RAX to Windows 2000, is to keep a separate count of unsuccessful password attempts for each user name. RAX would report unsuccessful password attempts to the system operator (a full-time employee, since RAX ran on a traditional IBM mainframe) and also report the number of bad password attempts each time the user successfully logged on. Windows 2000 provides a mechanism to explicitly limit password guessing: Windows will "lock out" an account and not accept any attempts to log on following an excessive number of unsuccessful attempts. The threshold is established by an administrator. Unfortunately, this lockout mechanism has a negative impact on usability as discussed in Chapter 6. ***D-4***

SOCIAL ENGINEERING

Some attackers don't even bother with password guessing; they simply ask for the password. This remains the biggest problem with passwords: there is no way to prevent someone's sharing secret information with someone else. Often, a clever attacker can trick someone into sharing a secret password, saving the attacker the trouble of performing a technical attack on the target system. Such trickery is usually called a *social engineering* attack.

The typical attack has been summarized best by Jerry Neal Schneider, one of the earliest computer criminals on record, who went into the computer security business after being released from jail. When interviewed in 1974, Schneider declared that he could break into any timesharing system in existence. He demonstrated this by cajoling a system operator into giving him a password. When the observer objected that Schneider hadn't "really" broken into the system, Schneider replied, "Why should I go to all the work of trying to technically penetrate a computer when it is so easy to con my way in through the timesharing service personnel?" ***A-6*** ***see Note 11.***

Another story, recounted by Katie Hafner and John Markoff, described a "cheerful technician" who called up a company, claimed to be the service representative for their computer vendor, and asked if they were having performance problems. Of course, just about everyone who owns a computer is convinced that it is not working as hard as it should, so the company was happy to give the technician a login and password so he could "fix" the problem. At some point, however, a site administrator found something suspicious. He called the vendor, only to find that no such "cheerful technician" worked there. ***see Note 12.***

Fortunately, most modern systems resist this attack because they use password hashing. Operators, administrators, and help desk people can't possibly reveal passwords because the password file contains only the hash values, not the password text. The remaining risk is the handling of lost passwords: an attacker could claim to be a legitimate user with a lost password. Then the attacker can talk the help desk into changing a victim's password and disclosing the new one to the attacker. The only way to protect against that is by restricting the password change procedure. Some sites might simply refuse to change users' passwords remotely, but instead require ***D-5*** that users arrange to replace lost passwords in person. Other sites might accept remote requests to change passwords, but then deliver the new password via a different (and hopefully safer) path that should reach the legitimate user instead of an attacker.

Of course, attackers can still social engineer a password without battling the help desk. One way is to approach individual users and trick them out of their passwords. For example, the attacker could call John and make a speech of the following sort:

> Mr. Doe, my security assessment has revealed that the symmetric bi-quinary azimuth of your datasets has been corrupted. You must log off immediately to prevent damage to your differential b-tree entries. I need to log on from your precise execution context in order to verify the longitudinal redundancy of your i-nodes. Please supply me with your most recent login name and password so I can prevent the irretrievable loss of your files.

While many computer users might recognize such a speech as pure drivel, others will be chanting their password before the attacker has run out of breath. Telephone fraudsters have used similar speeches for years to trick people out of credit card numbers. The successful attacker simply needs to convince the victim that disaster is imminent and the attacker can prevent it, once the victim divulges the secret.

The best defense against such an attack is to establish a policy of never, never sharing passwords with anyone, including administrators. Otherwise, an attacker can probably trick a gullible person into revealing a password simply by twisting the policy to make it sound like a legitimate request. Modern server systems should never require users to share their passwords with administrators. If administrators need access to something, they can generally do it from an administrative role. ***D-6***

A different attack yields a similar result by turning the tables. In April 1991, several Internet sites reported that their users received the following e-mail message, or variants thereof:

```
To: [ adddress list suppressed]
From: root
Subject: Security Alert

This is the system administration:
  Because of security faults, we request that you change
your password to “systest001”. This change is MANDATORY
and should be done IMMEDIATLY. You can make this change
by typing “passwd” at the shell prompt. Then, follow the
directions from there on.
  Again, this change should be done IMMEDIATLY. We will
inform you when to change your password back to normal,
which should not be longer than ten minutes.

               Thank you for your cooperation,

               The system administration (root)
```

see Note 13.

The message was, again, complete nonsense. The message took advantage of the ease with which attackers can forge Internet e-mail addresses in order to trick people into changing their passwords. Naturally, the attackers would simply wait for people to change their passwords, and somehow never get around to sending a message telling them to change their passwords back. ***A-7***

In a truly extreme case, an attacker might go after the victim directly and use threats or physical harm to extract a password or a similar secret. Indulging in black humor, cryptographers often refer to this as *rubber hose cryptanalysis*, since the secret could be an encryption key as well as a computer password. ***A-8***

One approach to reduce the risk of such attacks, particularly when the victim and proprietor are both trying to defend against the attack, is to implement a *duress signal* in the authentication mechanism. The signal is similar to the silent alarm a teller might activate during a bank robbery. In an authentication mechanism, the victim sends the signal via a seemingly legitimate variant of the conventional login procedure. For example, a person could have two pass- ***D-7*** words, one to indicate a legitimate login operation and a separate one to use when forced.

A duress signal, however, is only worthwhile if the victim is going to use it. Victims may be too traumatized to remember the duress signal, particularly if they've had no practice using it. Moreover, bank tellers use silent alarms when they believe it will increase their chances of survival. Few victims use a duress signal simply to protect the proprietor; they are more likely to use it if the signal will summon help but won't put them in greater danger.

In his role working with agents sent behind enemy lines during World War II, Leo Marks became very skeptical of the value of duress signals in real-world applications. In many cases, the agents appeared to have revealed all of their operating procedures to their captors, including signals for both routine and duress messages. Moreover, duress signals were rarely recognized as such, and the headquarters in Britain often interpreted them as erroneous transmissions. *see Note 14.*

A victim has to know exactly how the duress signal will be handled in order to have confidence in its value. At a minimum, a computer-based duress signal should alert the system's proprietor that a member of the user population is in serious trouble. The computer system itself should also take some action to minimize damage from the attack. However, victims are unlikely to use a duress signal if it simply disables their computer account, since that would clearly announce to the attacker that a duress signal was used.

A more promising approach is for the duress signal to cause subtle but important changes in the victim's capabilities as a system user. The attackers should feel that they have gained access to the victim's resources while really being prevented from doing serious harm to the system. Some sites use a similar approach for handling remote intrusions: the attackers are diverted to a special system called a *honey pot.* To the attackers, the honey pot looks like an attractive system to attack, when in fact its resources are all a sham to divert attackers' interest away from the really important systems. The honey pot is supposed to keep the attackers connected to the system while the proprietor's investigators, and possibly law enforcement, try to track them down. This approach could also be applied to duress signals: the signal would connect the victim to a *D-8* honey pot, giving the attackers the illusion of penetrating the system without actually placing critical resources at risk.

1.5 Sniffing Attacks

In password *sniffing*, the attacker tries to intercept a copy of the secret password as it travels from its owner to the authentication mechanism. Like guessing attacks, sniffing attacks start off with relatively trivial, low-tech mechanisms. But over the years, attackers have developed numerous high-tech approaches to sniffing. Risks from sniffing place a limit on all password systems: the more the password must travel around, the more opportunity attackers have to sniff it.

One type of sniffing starts right at a person's elbow: a nearby attacker can look over a person's shoulder and watch him type his password. This is called *shoulder surfing*. Modern systems either don't echo the password or they use a variant of the password blinding shown in Figure 1.4 to reduce this risk. Some systems go so far as to echo a different number of blinded characters than the actual number of characters typed. Despite these techniques, it is sometimes possible to watch the keystrokes themselves. Passwords are vulnerable to even more sniffing as they travel from the keyboard to the authentication mechanism.

A-9

D-9

SNIFFING IN SOFTWARE

Figure 1.7 shows a classic sniffing attack associated with timesharing systems, like the RAX system. When people logged in, the system copied the user name and password into a special area of RAM called the keyboard input buffer. The password checking procedure compared the data in the input buffer against the password stored on the RAX hard drive and logged the user on if the password matched. Meanwhile, however, another user on the RAX system could run a "sniffer" program that copied information out of the keyboard input buffer as people typed. The attacker watched the typed information especially closely when people logged on, since both the user name and the password would promptly show up in the input buffer.

A-10

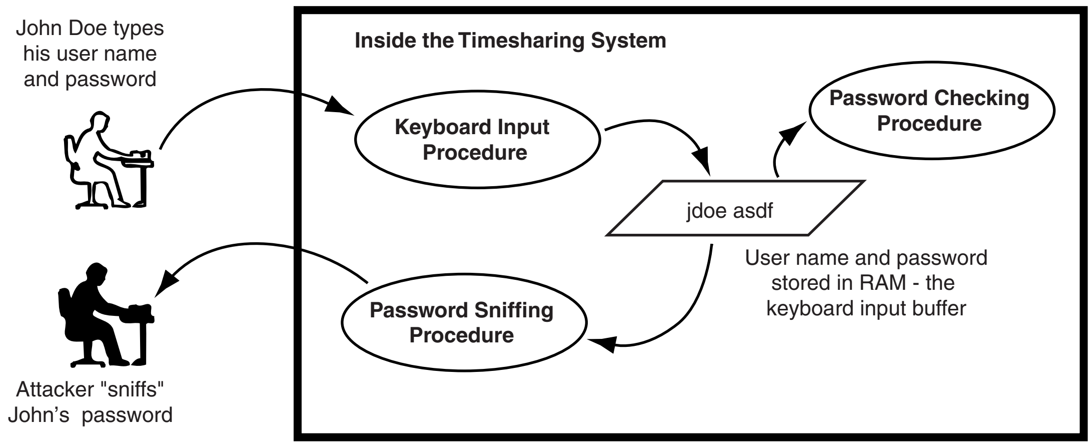

John Doe types his user name and password
Inside the Timesharing System
Keyboard Input Procedure
Password Checking Procedure
jdoe asdf
User name and password stored in RAM - the keyboard input buffer
Password Sniffing Procedure
Attacker "sniffs" John's password

FIGURE 1.7: *Password sniffing on a timesharing system.* John Doe's user name and password are read into RAM. The attacker runs a "sniffing" program that retrieves passwords from the RAM location where the keyboard input procedure reads them in. Password hashing does not protect against this attack, since the password must always arrive from the keyboard input procedure in plaintext form.

Although the RAX system at the time did not implement any form of password hashing or encryption, such a mechanism would not have protected against this attack. The keyboard input procedure had to read the entire password into RAM without hashing it first. RAX ran on an IBM mainframe (the IBM 360 family) and the IBM computer hardware put restrictions on how data was exchanged with peripheral devices like keyboards. In particular, the keyboard input procedure was written as a "channel program" that could only read a full line of text at a time from a keyboard. Whenever a user typed a password, the channel program had to read the entire line of text containing the password directly into RAM. There was no way for the RAX software to disguise the password before it was stored in RAM in a sniffable form.

Instead, the attack was blocked by changing the RAX system to restrict use of the keyboard input buffer. The IBM mainframe included memory protection features that RAX already used to keep user programs from accidentally writing information into other people's keyboard input buffers or from damaging other system data in RAM. The software was changed so that only the RAX system software itself was allowed to see the contents of people's keyboard input buffers. When the attacker's sniffer program attempted to **D-10** read data from someone's buffer, the mainframe hardware blocked the attempt, generated a signal, and RAX aborted the sniffer program.

Modern systems can be vulnerable to the same threat and usually solve it the same way. The hardware protections provided on classic IBM mainframes are standard equipment on state-of-the-art microprocessors like the PowerPC and the Pentium. Sophisticated operating systems like Unix, Mac OS X, Windows NT, and Windows 2000 protect keyboard input buffers from sniffing.

Unfortunately, traditional desktop systems like Windows 95 or 98, and single-user Macintosh systems, rarely make use of hardware protection mechanisms. Sniffers designed for single-user systems usually save the password information in a file or transmit it across a network to another computer. Sniffer programs targeting those systems are common features in collections of hacker tools. Well-known sniffers in the Windows environment include "keycopy," "playback," and "keytrap." Sniffer software poses a very real threat in academic and educational settings where roomfuls of personal workstations are shared among many students and occasionally with administrators.

Not all sniffer problems are hacker based. In an attempt to prevent computer misuse by employees, several organizations use *keystroke monitor* software. This software works almost exactly like a sniffer except that it collects *every* keystroke. This allows managers to review employee activities and identify employees that indulge in personal or recreational activities on company time or equipment. Unfortunately, any employee that looks at the keystroke monitor logs will be able to retrieve passwords of monitored employees. This makes the logs even more sensitive than system password files, since modern password files are usually hashed. Keystroke monitoring poses a dilemma wherever passwords are used, since managers must choose between monitoring and the reliable authentication of employees. ***see Note 15.***

Just as timesharing and other forms of distributed computing brought the need for authentication and passwords, it also produced some new threats as computing became even more distributed. As connections became more widespread and systems were used for more and more important tasks, sites had to contend with sniffing threats on their communications lines. Matters got even worse with the evolution of the Internet.

In early 1994, network news broadcasts made an unprecedented announcement: All computer users on the Internet should change their passwords. The announcement was triggered by a notorious sniffing incident: a centrally located Internet service had been penetrated and an attacker had installed unattended sniffer programs to collect passwords from the user traffic passing through it. Further investigations found sniffer programs in many other Internet sites. Investigators estimated that at least 100,000 passwords were collected by the sniffers they found. ***see Note 16.***

The evening news recommendation reflects a major advantage of passwords—if the user thinks the password has been stolen, it is relatively easy to block the attacker from exploiting the stolen one by replacing it with a new one. This is another essential defense: every password system must have a mechanism to allow users to change their passwords easily. ***D-11***

TROJAN LOGIN

Another clever method for intercepting passwords is a *Trojan login* program. This is a program that tricks people into revealing their passwords. Unlike the Trojan horse program described earlier, this one doesn't need to have the victim start it up. Instead, the attacker starts up the Trojan login program on some workstations or terminals, and the program's display perfectly mimics the system's standard login program. For example, a RAX Trojan login program would print the following text, just as shown in Figure 1.4: ***A-11***

```
#*rax v3 m86, sign on.
```

When someone like John Doe tries to log on by typing the appropriate command ("/id"), the Trojan login program saves a copy of his user name in a secret file. Then the program types "*Password?" on one line and the blinding characters on the next. When John types his password, the program responds by typing "*Password incorrect." Meanwhile, the program saves John's typed-in password in the file along with his user name. Finished, the program exits and logs out. If John tries again, he sees the same printout on the terminal, but this time it comes from the actual RAX login procedure. Meanwhile, the attacker has captured John's password.

Modern systems solve this problem with a *secure attention* signal. This is a special keystroke that the underlying system always recognizes as a request for special services like logging on. Whenever someone types that keystroke, the system intercepts it and runs a built-in program to find out what the person wants to do. The secure attention key provides a *trusted path* between the person operating the computer and a piece of trustworthy software on the computer. The U.S. government requires the presence of a trusted path mechanism on systems with extremely high security requirements as described in the TCSEC.

D-12

see Note 17.

User application programs, like the Trojan login program, never see the secure attention signal, so they can't perform their masquerade. Microsoft Windows NT and Windows 2000 use the keystroke control-alt-delete as a secure attention signal. Other systems, like Digital's VAX/VMS and high security versions of Unix, also provide secure attention keys.

VAN ECK SNIFFING

In 1985, a Dutch scientist named Wim van Eck described how one could eavesdrop on any video monitor using relatively simple techniques. Video monitors use a lot of energy, and the process of scanning data onto the phosphorescent screen of a video tube generates lots of stray electromagnetic signals. The signals are called *van Eck radiation* and, in theory, are visible from as far away as 1 kilometer.

see Note 18.

An attacker with the right equipment could read passwords and other secrets displayed on any nearby video screens. Win Schwartau, a well-known figure in information warfare circles, demonstrated the technique on a television show in 1991. Schwartau has suggested that a properly designed van Eck receiver would not be limited to intercepting video signals. In one theoretical scenario, a receiver could retrieve enough information from an automated teller machine (ATM) to reconstruct customers' bank cards and the corresponding personal identification number (PIN).

A-12

see Note 19.

The first defense against van Eck interception is to be sure that passwords do not appear on video displays. Most systems include this feature already, even though it has a serious impact on reliability and usability of the user interface (see Section 6.1). Additional protection could be provided by adding some shielding to video displays and other computer equipment. Unfortunately, research and engineering in this area traditionally has been discouraged in the United States by government intelligence agencies, notably the National Security Agency (NSA).

Classic passwords are not, of course, the only authentication technique available on computers. They are merely the oldest and easiest to implement. In the years since timesharing first evolved, password systems have improved dramatically and have incorporated additional security measures. Moreover, other authentication techniques have evolved to handle situations where passwords simply can't do the job.

1.6 AUTHENTICATION FACTORS

Things you know...
Things you have...
Things you are...

— Carlton et al., *Alternate Authentication Mechanisms*

Even before computers came along, people used a variety of distinguishing characteristics to authenticate one another. Computer systems have applied these characteristics whenever people have found a cost-effective way to implement them digitally. Today, authentication techniques are usually classified according to the distinguishing characteristic they use, and we classify the characteristics in terms of three *factors* described below and summarized in Table 1.2. Each factor relies on a different kind of distinguishing characteristic to authenticate people. ***see Note 20.***

- **Something you know: a password**

  The distinguishing characteristic is secret information that unauthorized people do not know. Before computers, this might be a spoken password or a memorized combination for a lock. In computers it might be a password, a passphrase, or a PIN.

  Developers can implement a plausible looking password mechanism cheaply and easily. A memorized secret is perfect for

TABLE 1.2: *Authentication Factors*

Factor	Benefits	Weaknesses	Examples
Something you know: password	Cheap to implement, portable	Sniffing attacks, Can't detect sniffing attacks, Passwords are either easy to guess or hard to remember, Cost of handling forgotten passwords	Password, PIN, Safe combination
Something you have: token	Hardest to abuse	Expensive, Can be lost or stolen, Risk of hardware failure, Not always portable	Token, Smart card, Secret data embedded in a file or device, Mechanical key
Something you are: biometric	Easiest to authenticate with, portable	Expensive, Replay threats, Privacy risks, Characteristic can't be changed, False rejection of legitimate users, Characteristic can be injured	Fingerprint, Eye scan, Voice recognition, Photo ID

roaming users, that is, people who connect to the system from unpredictable remote locations, since it travels with them.

Passwords are weak, however, for two reasons. First, their effectiveness depends on secrecy and it is hard to keep them secret. There are countless ways to sniff or otherwise intercept them, and there is usually no way to detect a successful sniffing attack until damage is done. Second, evolving threats on passwords have made it relatively easy for attackers to figure out the passwords that people are most likely to choose and remember. Even if they choose hard-to-guess passwords, people are more likely to forget them or be obliged to write them down in order to have them available when needed. A written password is, of course, more vulnerable to theft than a memorized one. Even well-meaning people are likely to violate password usage rules at some point, simply to ensure they can use their computer when needed. Chapters 2 and 3 discuss passwords further, and Chapter 6 explores the problem of choosing effective passwords.

- **Something you have: a token**

  The distinguishing characteristic is that authorized people possess some specific item. Before computers this might be a seal with a personal insignia or a key for a lock. In computers it might be nothing more than a data file containing the distinguishing characteristic. Often, the characteristic is embedded in a device like a magnetic stripe card, a smart card, or a password calculator. In this book, such things are called *tokens*. The characteristic might even be embedded in a large piece of equipment and thus not be very portable.

Token-based authentication is the hardest technique to abuse since it relies on a unique physical object that one must have in order to log on. Unlike passwords, the owner can tell if the token has been stolen, and it's hard for the owner to share the token with others and still be able to log on. The major weaknesses are higher costs and the risk of loss or hardware failure. Portability can also be a problem. Tokens are discussed further in Chapter 9.

- **Something you are: a biometric**

  The distinguishing characteristic is some physical feature or behavior that is unique to the person being authenticated. Before computers, this might have been a personal signature, a portrait, a fingerprint, or a written description of the person's physical appearance. With computers, the person's distinguishing characteristic is measured and compared against a previously collected pattern from the authentic person. Well-known techniques use a person's voice, fingerprints, written signature, hand shape, or eye features for authentication. In this book, such things are called *biometrics*.

  Biometric authentication is usually the easiest approach for people to use for authentication. In most cases, a well-designed biometric system simply accepts a reading from the person and correctly perform the authentication. The distinguishing characteristic is obviously portable, since it's part of the owner's body.

  However, the benefits are offset by several weaknesses. Typically, the equipment is expensive to buy, install, and operate in comparison to other systems. Biometric readings face the risk of interception when used remotely; the thief might replay the reading to masquerade as its owner or use the reading to track its owner. Once the biometric reading has fallen into the wrong

> hands, its owner has no way to reverse the damage, since the biometric trait is usually impossible to change.
>
> Moreover, the process is a tricky one. In practice, it can be hard to make the system sensitive enough to reject unauthorized users without occasionally rejecting authorized users. Physiological changes and injuries can also invalidate biometric readings: in one case a woman working at a high-security installation was denied entrance by the biometric device at the front door because her pregnancy had caused changes in her retinal blood vessels.
>
> Despite their shortcomings, biometrics remain a promising technique. Biometrics are discussed further in Chapter 7.

In other words, authentication always depends on something lost, injured, or forgotten. There really is no "one best way" to authenticate people. The choice depends on the particular risks faced by a computing system and the costs (in terms of equipment, administration, and user impact) that the proprietor is willing to incur. Sites often rely on passwords because of lower costs: the implementation requires no special hardware to purchase, install, and maintain. Organizations use other techniques only when the potential loss from mishandled passwords clearly exceeds the cost of something different.

All authentication factors have their own shortcomings, and individual factors can't always provide the level of protection a site might need. In such cases, sites will use authentication mechanisms that incorporate two or three factors. Such systems are often referred to as *strong authentication* since the benefits of one factor can block the shortcomings of another. ATM cards, which always require a memorized PIN, provide a well-known example of two-factor authentication.

A common theme in all three factors is that the distinguishing characteristic is uniquely associated with the person being authenticated. This becomes a problem when operating over computer networks. Often, the authentication mechanism has nothing to go on except a collection of bits derived from the distinguishing characteristic. There is no mechanism (aside from additional authentication) to establish the provenance of the bits themselves. There is no way to tell if a biometric reading was really collected from the person at the other end or if it was sniffed and retransmitted by someone else. There is no way to tell if the password comes from the authorized person or from someone else. In short, simple authentication mechanisms often rely on keeping the distinguishing characteristic secret.

1.7 JUDGING ATTACK PREVALENCE

While it is important to be aware of the attacks that could take place against a given security technique, it is also important to understand how likely a particular attack might be. This often depends on the degree of sophistication each attack requires. Most proprietors won't be as concerned about attacks that require the resources of a national government as they are about an attack that's known by every teenaged Internet user.

This book classifies attacks into five levels of prevalence. The different levels indicate the relative knowledge and resources the attacker will need, as well as the degree to which such attacks seem to occur. Attackers faced with very attractive targets will spend significant amounts of effort on intrusions. The rest of this book uses these classifications to indicate the prevalence of attacks.

- **Trivial attack**

  There are books filled with trivial attacks. Anyone who knows the "trick" behind one of these attacks can perform it using conventional software already present on a typical workstation. Internet e-mail forgery is the classic example of a trivial attack. Anyone with a typical Internet e-mail package on a personal computer can configure it to generate e-mail that claims to be from someone else (for example, president@whitehouse.gov). The attack doesn't rely on special hacker tools or nonstandard software. Anyone can do it if he or she knows how. ***see Note 21.***

- **Common attack**

  Unlike trivial attacks, the attacker must acquire specific software tools or take similar steps that may indicate premeditation. Password sniffing is an example of a common attack: a hacker installs keystroke capture software that collects someone's password to exploit later. Viruses are another example: the author either must use a virus construction program to create the virus or must intentionally write the virus from scratch. The presence of attack software on workstations indicates the owners may be either victims or perpetrators of software-based attacks, depending on the particular software involved. Often, there are well-known security measures to apply to common attacks. Unfortunately, there are also a large and growing number of common attacks. There are tools that help attackers detect well-known vulnerabilities and it can be time consuming to try to block all of them.

- **Physical attack**

  These attacks require the attacker's physical presence at the point of attack. They may also require the manipulation of computer hardware and/or special hardware tools. The attacks may also depend on special knowledge and training. In Section 1.4 we briefly described "rubber hose" attacks, but the other physical attacks in this book are directed against equipment, not people. In Chapter 4 we discuss attacks in which the attacker opens the computer case and rewires the hard drive in order to copy its contents. Other examples include sniffing or wiretapping when the connecting wire is physically tapped. Unlike common attacks, physical attacks tend to leave physical evidence, and this tends to deter less motivated attackers.

- **Sophisticated attack**

  A sophisticated attack is an attack that requires sophisticated knowledge of security vulnerabilities. While common attacks may be performed by people with good tools and limited knowledge or skill, a sophisticated attack may require the attacker to construct a tool to implement the attack. The work of the Wily Hacker tracked by Clifford Stoll in the late 1980s is an example of a sophisticated attack: the hacker had a "bag of tricks" consisting of trivial and common attacks, and he applied multiple tricks to penetrate and exploit the systems he attacked. The attacks and exploitation took a significant effort by an individual attacker working part-time. Sites defend against sophisticated attacks by restricting their operations to reduce their vulnerability to them.

see Note 22.

- **Innovative attack**

  These are attacks that exploit theoretical vulnerabilities that have not been publicly demonstrated to be practical. Such attacks often bring significant resources to bear in order to breach a strong security mechanism. An innovative attack probably involves a large, well-funded team of attackers working for months or years to breach the adversary's defenses. For example, Allied cryptanalysts during World War II made an innovative attack against German cryptographic equipment by combining mathematical analysis with trial-and-error decryption attempts. Innovative attacks tend to be very expensive, but they also tend to succeed if the attacker is willing to invest the resources needed to achieve success. Nuclear command and control systems are designed to resist innovative attacks by spending enormous amounts of money on redundancy and least privilege. *see Note 23.*

While security experts may often agree in general terms about the prevalence of various attacks, there is no widely accepted notion of attack prevalence or how to estimate it. The estimates of prevalence appearing in this book are based on attack reports and descriptions, and try to reflect the consensus of the computer security community. These estimates primarily reflect the author's qualitative opinion and are not based on any systematic measurements. As time goes on and attacks evolve, the prevalence of various attacks will no doubt change.

1.8 Summary Tables

The last section of each chapter contains two summary tables: one summarizing attacks described in the chapter, and another describing the corresponding defenses. The attack summary gives a brief name for each attack, summarizes the security problem, estimates the sophistication, and briefly describes how the attack works. Sophistication is assessed using the classification described in Section 1.7. If modern software systems routinely provide a defense against a particular attack, it is marked as "Obsolete" in the attack summary. The defense summary gives a brief name for each defense, a list of attacks it should protect against, and a brief description of what it does. If the chapter introduces an attack for which it introduces no defense, the attack is noted in an additional subsection titled **Residual Attacks**.

TABLE 1.3: *Attack Summary*

Attack 💣	Security Problem	Prevalence	Attack Description
A-1. Keystroke confusion	Masquerade as someone else	Obsolete	A bug found in the timesharing software allowed a peculiar sequence of characters to skip password checking
A-2. Password file theft	Recover all other users' passwords	Obsolete	Weak protection of password file allowed its contents to be stolen
A-3. Trojan horse	Recover hidden information, like a password file	Common, Sophisticated, or Innovative	Attacker writes a program that gets used by the victim. Unknown to the victim, the program copies or modifies the victim's data. A virus is a well-known example
A-4. On-line password guessing	Recover a user's password	Trivial	Interactive trial-and-error attack to try to guess a user's password
A-5. Password audit review	Recover a user's password	Common	Review audit records of a user's mistakes while logging on to make guesses of the user's password
A-6. Helpful disclosure	Recover a user's password	Trivial	Attacker convinces a victim to reveal a password in support of an apparently important task
A-7. Bogus password change	Recover a user's password	Trivial	Attacker convinces victims to change their passwords to a word selected by the attacker
A-8. Rubber hose disclosure	Recover hidden information, like a user's password	Physical	Attacker uses threats or physical coercion to recover secret information from the victim
A-9. Shoulder surfing	Recover a user's password	Trivial	Attacker watches a user type his password, then uses it himself

TABLE 1.3: *Attack Summary (Continued)*

Attack	Security Problem	Prevalence	Attack Description
A-10. Keystroke sniffing	Recover a user's password	Common	Software watches keystrokes transmitted from the user to the system for typed-in user names and passwords, save for later use
A-11. Trojan login	Recover a user's password	Common	Run a program that mimics the standard login program, but collects user names and passwords when people try to log on
A-12. van Eck Radiation	Recover hidden information, like a user's password	Physical	Use a device to intercept van Eck radiation from the victim's video monitor, and retrieve any secrets the victim displays

TABLE 1.4: *Defense Summary*

Defense	Foils Attacks	Description
D-1. Good software design	A-1. Keystroke confusion	Design software in an organized way to reuse existing functions; keep procedures simple and comprehensible
D-2. Hashed passwords	A-2. Password file theft A-6. Helpful disclosure	Store passwords in a one-way hashed format. Avoid storing or handling the password in its readable, unhashed form
D-3. Audit bad passwords	A-4. On-line password guessing	Keep an audit trail of all attempts to log on, and use the trail to detect password guessing attacks
D-4. Limit password guessing	A-4. On-line password guessing A-5. Password audit review	Keep track of the number of incorrect guesses someone may make of a password, and respond to excessive guesses as indicating a password guessing attack

TABLE 1.4: *Defense Summary (Continued)*

Defense	Foils Attacks	Description
D-5. Password change policy	A-6. Helpful disclosure	Establish a policy to restrict the ability of the help desk to change passwords for users. Password changes must take place under safe circumstances
D-6. Password nondisclosure policy	A-6. Helpful disclosure A-7. Bogus password change	Establish a policy that nobody should disclose a password to another person under any circumstances
D-7. Duress signal	A-8. Rubber hose disclosure	Lets a user signal that the login process is taking place under duress
D-8. Honey pot	A-8. Rubber hose disclosure	Allows attackers to enter the system, presents them with a legitimate-appearing target, while restricting their access to truly valuable resources and keeping them under surveillance
D-9. Password blinding	A-9. Shoulder surfing	Do not print or display the keys typed when the user types a password
D-10. Memory protection	A-10. Keystroke sniffing	Use the CPU's memory protection feature to protect the keyboard input buffer from reading by any software except the OS
D-11. Change password	A-2. Password file theft A-9. Shoulder surfing A-10. Keystroke sniffing	The password's owner can change the password to something new when there is a risk that it has been intercepted by an attacker
D-12. Secure attention	A-11. Trojan login	System assigns a special keystroke to security-related user requests like logging on

RESIDUAL ATTACKS

A-3. Trojan horse—There is no general-purpose defense against this attack. There are techniques to resist Trojan horses designed to operate in particular ways. For example, antivirus software addresses the virus problem. There are also integrity checking programs that can detect modifications to files that rarely change, like the login program.