5

Authentication

“Who are you, Master?” he asked.

“Eh, what?” said Tom sitting up, and his eyes glinting in the gloom. “Don’t you know my name yet? That’s the only answer. Tell me, who are you, alone, yourself and nameless.”

Lord of the Rings
—J.R.R. TOLKIEN

Authentication is the process of *proving* one’s identity. This is distinct from the assertion of identity (known, reasonably enough, as *identification*) and from deciding what privileges accrue to that identity (*authorization*). While all three are important, authentication is the trickiest from the perspective of network security. We are concerned here with two forms of authentication, that of a user to a machine during an initial login sequence, and machine-to-machine authentication during operation. Solutions to the first problem are typically categorized as “something you know,” “something you have,” and “something you are.” Machine-to-machine authentication is generally divided into two types: “cryptographic” and “other.” (Some would say “cryptographic” and “weak.”)

The level of authentication depends on the importance of the asset, and the cost of the method. Should you use hand-held authenticators for logins from the outside? What about from the inside? What sort of authentication do you want for outgoing calls? How about privileged (*root*) access to machines? For that matter, who will maintain the firewall’s authentication databases? In many environments, a single firewall serves the entire organization, which makes tracking personnel that much harder. Would a scheme that does not require databases be better? Many options are listed in this chapter.

5.1 User Authentication

5.1.1 Passwords

There is little more to say here about passwords; we already discussed it at some length in Section 1.3.3 and will discuss it further in 9.1. As a means of personal authentication, passwords are categorized as “something you know.” This is an advantage because no special equipment is required to use them, and also a disadvantage because what you know can be told to someone else, or guessed, or captured.

No security expert we know of regards passwords as a strong authentication mechanism. Nevertheless, they are unlikely to disappear any time soon, because they are simple, cheap, and convenient.

5.1.2 One-Time Passwords

One can achieve a significant increase in security by using *one-time passwords*. A one-time password behaves exactly as its name indicates: It is used exactly once, after which it is no longer valid. This provides a very strong defense against eavesdroppers, compromised *telnet* commands, and even publication of login sessions. (See Figure 5.1 on page 121 for an example of the latter.)

There are a number of possible ways to implement one-time password schemes. The best known involve the use of some sort of *hand-held authenticator* , also known as a *dongle* or a *token*.

One common form of authenticator contains an internal clock, a secret key of some sort, and a display. The display shows some function of the current time and the secret key. This output value, which is used as the authentication message, changes about once per minute. (The use of cryptography to implement such functions is described in Chapter 13.) These “passwords” are never repeated.

The host validates the user by using its copy of the secret key and its clock to calculate the expected output value. If they match, the login is accepted. In practice, clock skew between the device and the host can be a problem. To guard against this, several candidate passwords are computed, and the user’s value is matched against the entire set. A database accessible to the host keeps track of the device’s clock skew to help minimize the time window. But this introduces another problem: A password could be replayed during the clock skew interval. A proper implementation should cache all received passwords during their valid lifetime; attempted reuses should be rejected and logged.

A different one-time password system uses a nonrepeating challenge from the host instead of a clock. Again, the user has a device that is programmed with a secret key. The challenge is keyed into the device, which calculates some function of it and the key, and this value serves as the password. Since there is no clock involved, there is no clock skew, and hence no need for a cache. On the other hand, the device must have a keypad, and the user must transcribe the challenge. Some have complained about this extra step. A sample *telnet* session through our gateway is shown in Figure 5.1.

```
$ telnet guard.research.att.com
Trying...
Connected to guard.research.att.com.
Escape character is '^]'.

This is the new inet. Authorized use only.


Authentication Server.

Id? ches
challenge: 48201
response: d2c3f97d


TCP host name? cetus
rlogin cetus '-e' -8 -l ches

IRIX Release 4.0.5C System V cetus.research.att.com
Copyright 1987-1992 Silicon Graphics, Inc.
All Rights Reserved.
cetus=; exit
Connection closed.Connection closed by foreign host.
$
```

Figure 5.1: The full text of an actual terminal session using our challenge/response-based guard. The challenge is random, and will rarely, if ever, be repeated. An enemy cannot replay an old response, because it is derived from the challenge, which should be different for each session. The parameters on the *rlogin* command prevent a shell escape to the gateway machine. A modified *rlogin* would probably be better.

Challenge/response identification is derived from the *Identification Friend or Foe* (*IFF*) devices used by military aircraft [Diffie, 1988]. It, in turn, is derived from the traditional way a military sentry challenges a possible intruder.

Both of these schemes involve "something you have," a device that is subject to theft. The usual defense is to add "something you know" in the form of some sort of *personal identification number* (*PIN*). An attacker would need possession of both the PIN and the device to impersonate the user. (Note that the PIN is really a password used to log in to the hand-held authenticator. Although PINs can be very weak, as anyone in the automatic teller machine card business can testify [Anderson, 1993], the combination of the two factors is quite strong.) Also, either approach must have the key accessible to the host, unless an authentication server is used. The key database can be a weakness and must be protected.

Many people carry a computer around these days. These algorithms, and especially the following, are easily implemented in a portable machine.

Lamport proposed a one-time password scheme [Lamport, 1981] that can be implemented without special hardware. Assume there is some function F that is reasonably easy to compute in the forward direction but effectively impossible to invert. (The cryptographic hash functions described in Section 13.1.7 are good candidates.) Further assume that the user has some secret—perhaps a password—x. To enable the user to log in some number of times, the host calculates $F(x)$ that number of times. Thus, to allow 1000 logins before a password change, the host would calculate $F^{1000}(x)$, and store only that value.

The first time the user logs in, he or she would supply $F^{999}(x)$. The system would validate that by calculating

$$F(F^{999}(x)) = F^{1000}(x).$$

If the login is correct, the supplied password—$F^{999}(x)$—becomes the new stored value. This is used to validate $F^{998}(x)$, the next password to be supplied by the user.

The user's calculation of $F^n(x)$ can be done by a hand-held authenticator or a trusted workstation or portable computer. Bellcore's implementation of this scheme [Haller, 1994], known as *S/Key*, goes a step further. While logged on to a secure machine, the user can run a program that calculates the next several login sequences, and encodes these as a series of short words. A printed copy of this list can be used while traveling. The user must take care to cross off each password as it is used. To be sure, this list is vulnerable to theft, and there is no provision for a PIN. S/Key can also run on a PC.

An implementation of S/Key as part of a full firewall configuration is described in [Avolio and Ranum, 1994].

5.1.3 Smart Cards

A *smart card* is a portable device that has a CPU, some input/output ports, and a few thousand bytes of nonvolatile memory that is accessible only through the card's CPU. If the reader is properly integrated with the user's login terminal or workstation, the smart card can perform any of the validation techniques just described, but without their weaknesses. Smart cards are "something you have," though they are often augmented by "something you know", a PIN.

Some smart cards have hand-held portable readers. Some readers are now available in the PCMCIA format.

Consider the challenge/response scheme. As normally implemented, the host would need to possess a copy of the user's secret key. This is a danger; the key database is extremely sensitive, and should not be stored on ordinary computers. One could avoid that danger by using public-key cryptographic techniques (Section 13.1.4), but there's a problem: The output from all known public key algorithms is far too long to be typed conveniently, or even to be displayed on a small screen. A smart card, though, can not only do the calculations, it can transmit them directly to the host via its I/O ports. For that matter, it could read the challenge that way, too, and simply require a PIN to enable access to its memory.

5.1.4 Biometrics

The third method of authenticating a user attempts to measure something intrinsic to the user. This could be something like a fingerprint, a voiceprint, or a signature. Obviously, special hardware is required, which limits the applicability of biometric techniques to comparatively few environments. The attraction is that a biometric identifier can neither be given away nor stolen.

In practice, there are some limitations. Conventional security wisdom says that authentication data should be changed regularly. This is difficult to do with a fingerprint. Some methods have encountered user resistance; Davies and Price [1989] cite a lip-print reader as one example. Also, by their very nature, biometrics do not give exact answers. No two signatures are absolutely identical, even from the same individual, and discounting effects such as exhaustion, mood, or illness. Some tolerance must be built into the matching algorithm. Would you let someone log in if you were 93% sure of the caller's identity?

Some systems use smart cards to store the biometric data about each user. This avoids the need for host databases, instead relying on the security of the card to prevent tampering. It is also possible to incorporate a random challenge from the host in the protocol between the smart card and the user, thus avoiding replay attacks.

As of now, we are unaware of any routine use of biometric data on the Internet. But as microphone-equipped machines become more common, usage may start to spread.

5.2 Host-to-Host Authentication

5.2.1 Network-Based Authentication

For better or worse, the dominant form of host-to-host authentication on the Internet today relies on the network. That is, the network itself conveys not just the remote user's identity, but is also presumed to be sufficiently accurate that one can use it as an authenticated identity. As we have seen, this is dangerous. Network authentication itself comes in two flavors, address-based and name-based. For the former, the source's numeric IP address is accepted. Attacks on this form consist of sending something from a fraudulent address. The accuracy of the authentication thus relies on the difficulty of detecting such impersonations—and detecting them can be very hard.

Name-based authentication is weaker still. It requires that not just the address be correct, but also the name associated with that address. This opens up a separate avenue of attack for the intruder: corrupting whatever mechanism is used to map IP addresses to host names. The attacks on the DNS (Section 2.3) attempt to exploit this path.

5.2.2 Cryptographic Techniques

Cryptographic techniques provide a much stronger basis for authentication. While the techniques vary widely (see Chapter 13 for some examples), they all rely on the possession of some "secret" or cryptographic key. Possession of this secret is equivalent to proof that you are the party known to hold it. The hand-held authenticators discussed earlier are a good example.

If you share a given key with exactly one other party, and receive a message that was encrypted with that key, you *know* who must have sent it. No one else could have generated it. (To be sure, an enemy can record an old message, and retransmit it later. This is known as a *replay attack*.)

You usually do not share a key with every other party with whom you wish to speak. The common solution to this is a *Key Distribution Center* (*KDC*) [Needham and Schroeder, 1978, 1987; Denning and Sacco, 1981]. Each party shares a key—and hence some trust—with the KDC. The center acts as an intermediary when setting up calls. While the details vary, the party initiating the call will contact the KDC and send an authenticated message that names the other party to the call. The KDC can then prepare a message for the other party, and authenticate it with the key the two of them share. At no time will the caller ever learn the recipient's secret key. Kerberos (Section 13.2) is a well-known implementation of a KDC.

While cryptographic authentication has many advantages, a number of problems have blocked its widespread use. The two most critical ones are the need for a secure KDC, and the difficulty of keeping a host's key secret. For the former, one must use a dedicated machine, in a physically secure facility, or use a key exchange protocol based on public key cryptography. Anyone who compromises the KDC can impersonate any of its clients. Similarly, anyone who learns a host's key can impersonate that host and, in general, any of the users on it. This is a serious problem, since computers are not very good at keeping long-term secrets. The best solution is specialized cryptographic hardware—keep the key on a smart card, perhaps—but even that is not a guaranteed solution, because someone who has penetrated the machine can tell the cryptographic hardware what to do.