ELIOT LEAR

being awash in keys

A member of the USENIX community since 1988, Eliot Lear works for Cisco Systems, Inc., and has been their self-proclaimed corporate irritant since 1998. He's been a consumer a bit longer.

lear@cisco.com

TODAY'S CONSUMER MUST KEEP TRACK of many passwords in order to make use of online banking, commerce, and government services. Attempts to consolidate authentication methods into a single sign-on service have thus far failed. Worse, with viruses, spam, and phishing, more sophisticated authentication is demanded. That sophistication today has brought confusion to the consumer. What stops the consumer from having a single secure password? This article considers several areas of improvement the industry should consider, and we'll put forth an example of a single smartcard that could potentially provide unified authentication.

A (Very) Brief History of Success

The enterprise has largely been successful at consolidating identity. This evolution began with common host access through mechanisms such as Kerberos.[1] Common network access was then made possible through protocols and mechanisms such as TACACS and RADIUS.[2] Application integration is now following as directory services evolve. All of this is great for the enterprise, because with a single administrative function tied to a registrar or human resources function, it is possible to enable or disable a user, change user rights, and retrieve a log of user activity. This success has been made possible by a vested interest in consolidating administrative overhead costs. Unfortunately, not only are most of the mechanisms developed for the enterprise inappropriate for consumer authentication, but those products actually contribute to the individual's inconvenience. What does the consumer need? What is necessary for the e-commerce vendor?

The Phishing Example

Today most US banks use simple username/password security with one-way SSL authentication and encryption.[3] Put another way, the bank server authenticates itself to the user's browser, and in return the user sends a username and password through a form to authenticate himself or herself to the server. From a protocol standpoint, this method provides mutual authentication. It breaks down, however, at the user interface level. Published reports of US bank losses range from $500 million to $1.2 billion last year, while British banks lost over £1 billion.

In order to stem the losses, some banks have attempted to up the ante by requiring sophisticated challenge-response systems. Even these systems, however, are not impervious to attack. Consider the case in which the masquerader receives from a potential victim a username necessary for generating a challenge, sends the username to the real bank, parses the challenge, and then sends it back to the victim. In other words, these miscreants are able to effect a classical man-in-the-middle attack, all because the user didn't bother to click the little lock icon in his browser to verify that the certificate was correct. Worse, most people don't know what the correct certificate would look like.

This same bank might want to raise the stakes further by attempting to make the challenge readable only to humans (and perhaps just barely at that). Unfortunately, the same technology used by spammers and phishers is also used by legitimate screen readers for those who are visually impaired.

One approach to solve this problem is to make use of public/private key pairs in the smartcards. There are many methods to choose from, but the key point is finding where the line of trust begins and ends. SANS infection statistics claim that the average survival time of an unpatched Windows PC is 19 minutes.[4] Clearly the PC is not to be trusted. This means that the point of trust must be the smartcard itself, and that it must maintain an opaque channel through the PC to the authentication system on the other side.

Multiply the single bank login by investment houses, health insurance, travel sites, bookstores, telephone companies, not to mention an employer, so that in order to retain convenient access one needs a whole bag of hardware and a whole slew of login names. Instead, wouldn't it be nice if the user could make use of one or more identities to authenticate against any particular service?

Privacy concerns also abound. Any solution in this space will have to come to grips with the idea that vendors may wish to sell information about consumer identities. While some jurisdictions, such as the European Union, provide strong controls for consumers, others, such as the US, do not. Such correlation of information is difficult to prevent, in part thanks to HTTP and cookies. While it is unlikely that any solution in this space could help consumer privacy, one wishes not to add to the problem.

"Many Have Tried"

There have been several attempts at providing individuals with unique identities that could be used for commerce. The reasons they have failed are complex, including implementation limitations, trust of the provider, vendor costs, and competitive concerns. As we've seen, a software implementation leaves a tremendous amount to be desired. Any solution in this problem space must be widely accepted by both consumers and vendors, thus requiring that the needs of both be met. Inasmuch as credit card companies indemnify consumers from identity theft, they too have skin in this game.

There exist a number of standards in this space already. A plethora of ITU (International Telecommunication Union) standards define the interface between smartcards, computers, and identities. SASL, SSL, and TLS provide a means of transporting authentication over the network, and Mozilla provides a way to use hardware tokens. However, none of these standards has established a sufficiently trustworthy path between the smartcard and the server on the other end. In short, because everyone wants to be king of the mountain, nobody has been able to ascend.

Getting to a Secure Single Password

Reviewing our discussion, we can begin to see the form of a solution and can derive some requirements. Here, then, are mine:

- First, while a beautiful dream to some, a PKI deployed globally to all consumers has not happened yet, and there is no reason to believe conditions will change. Therefore, a solution should not rely on such a concept.
- No single vendor can own the market. Anyone playing King of the Mountain will be King of the Molehill. This implies use of open standards from the authenticator to the server, inclusively.
- The mechanism must be easy to use.
- The mechanism must handle multiple identities.
- The mechanism must be secure, not only from the network but from the host computer itself. This includes the computer keyboard!
- Finally, the mechanism must not cost an arm and a leg. We consumers are price-conscious!

With these requirements in mind, let's take a look at a straw man.

A Straw-Man Solution

What follows is not a complete answer to all consumer concerns. It is submitted for purposes of discussion.

Posit a smartcard with a small LCD display and a keypad (or other accessibility mechanisms) whose purpose is merely to provide mutual authentication for

any single transaction. The interface between this card and a host computer would likely be USB, due to power concerns.

Each service will have its own identity, which will contain the following fields:

- A name that is a randomly generated 2048-bit number
- A nickname supplied by the service
- A 128-bit serial number
- A 2048-bit public/private key pair

The card will have the following functions:

- `longlonglong createIdentity(char *nickname, int *serial, char *public)`
- `confirmIdentity(longlonglong name, char *cryptext)`

These functions are called not by the host computer but by the remote server. They are only accessible *if and only if* the user confirms them on the smartcard itself.

Let us assume that the identity with the nickname of "Joe Blow's Bookstore" exists with a name of N and a serial number of S. When I connect to Joe Blow's Web site and want to authenticate a transaction, it will call the function `confirmIdentity()`. `cryptext` will be encrypted in the previously generated public/private key pair and will contain the next expected serial number and a comment indicating the nature of the transaction. Assuming that there is such a name on my smartcard and the `cryptext` is decrypted properly, if the serial number is correct, both it and the comment will be displayed on the smartcard, along with the nickname. If all looks correct to me, I push the green button. If it looks incorrect, I either do nothing or push the red button. Similarly, the card should report all failed access attempts.

What has this accomplished? First and foremost, no more passwords are sent on the wire—encrypted or not. Second, I no longer rely on the host computer for security. Third, I may store as many identities as I wish in as many smartcards as I wish. Fourth, no individual vendor can share an identity in such a way that a third party could make use of it for purposes of authentication without me knowing about it. Finally, no remote server will be able to guess even the mere existence of other identities on the card. Other authorized parties may tell them about them, but if they try to access the identity, I'll know something fishy is going on. Note that X.509 certificates are not needed for the common usage case.

What happens if someone other than Joe Blow's Bookstore tries to use the nickname for Joe Blow's bookstore? When the user is asked to create the new identity when he is not at Joe Blow's Bookstore, he should notice that something is wrong. However, even if he does create a new identity that has the same nickname, all this phisher has access to is its own fictitious identity and not the real Joe Blow's, because the names will differ.

The tricky part is in `createIdentity`. As with any authentication system, the weakest part is always in the bootstrapping process. Here the risk is that somehow the computer or other device between the card and the authentication server might eavesdrop or otherwise perform a man-in-the-middle attack. In order to protect against such a thing, that path must be encrypted. For this use only, a certificate may be warranted. However, one needs signed certificates for the server and for the card, not for the individual. The server just needs to verify that the endpoint is a sufficiently secure card.

Another approach would be to use out-of-band information, although we run the risk of having the same accessibility problems mentioned earlier. For instance, the user could enter the ID, as well as the serial number, directly on the card. This method is cumbersome to the user.

The astute observer will note that there is no method to list identities. Such a function should be considered dangerous, because if the authorized user can execute it from the host computer, then so might someone else. If such a listing is displayed, it should be displayed on the card itself.

Conclusion

There are numerous problems with the straw man above. The goal of the exercise was to provide some idea of what standards are needed and which *aren't* needed (X.509 for most transactions), and to demonstrate the sort of user interface that is required. An open standard is needed to exchange authentication information all the way from the card to the authentication server and back again.

Enterprises have an interest in this sort of solution as well, since solving the problem in the consumer space brings with it the consumer market's economies of scale. Tokens are already pretty cheap. Having to manage them, however, has been another matter entirely. That, too, could be addressed with such a solution.

The hardest part of this problem remains the registration of identities. In this limited sense, use of a PKI may be justified.

One limitation of my approach is that it doesn't easily allow for consolidation of credit cards, which are nothing more than keys themselves. Because identities are kept secret on the smartcard and are selected by the authorizing party, there is no way for the user to specify authorization of a charge on a particular account.

NOTES

1. S. Miller et al., "Kerberos: An Authentication Service for Open Network Systems," *Proceedings of the USENIX Winter Conference1988 (*February 1988), pp. 191–202.

2. C. Rigney et al., "Remote Authentication Dial-In User Service (RADIUS)," RFC 2865 (June 2000); W. Yeong et al., "X.500 Lightweight Directory Access Protocol," RFC 1487 (July 1993).

3. A. Frier et al., "The SSL 3.0 Protocol," Netscape Communications Corp. (November 18, 1996).

4. The SANS Institute, "Survival Time History" (August 2004): http://isc.incidents.org.