Authentication

Patrick McDaniel

AT&T Labs - Research

May 31, 2002

OUTLINE

1 – Authentication
- Meet Alice and Bob

2 - Credentials

3 – Web Authentication
- Password Based Web Access
- Single Sign-on
- Certificates
- SSL

4 – Host Authentication
- Remote Login
- SSH
- One-Time Passwords
- Kerberos
- Pretty-Good-Privacy
- IPSec

5 - Conclusions

GLOSSARY

- Credential - evidence used to prove identity or access rights.
- Certificate - a digitally signed statement associating a set of attributes with a public key. Most frequently used to associate a public key with a virtual or real identity (i.e., identity certificate).
- Authentication - the process of establishing the identity of an on-line entity.
- Authorization - the process of establishing the set of rights associated with an entity.
- Trusted third party - an entity mutually trusted (typically by two end-points) to assert authenticity or authorization, or perform conflict resolution. Trusted third parties are also often used to aid in secret negotiation (e.g., cryptographic keys).
- Web of trust - self-regulated certification system constructed through the creation of ad hoc relationships between members of a user community. Webs are typically defined through the exchange of user certificates and signatures within the Pretty-Good-Privacy (PGP) system.
- Malicious party - entity on the Internet attempting to gain unauthorized access, disrupt service, or eavesdrop on sensitive communication (syn: adversary, hacker).
- Secret - information only known to and accessible by a specified (and presumably small) set of entities (e.g., passwords, cryptographic keys).

ABSTRACT

Authentication is the process by which the identity of an entity is established. Authenticating entities present credentials, such as passwords or certificates, as evidence of their identity. The entity is deemed authentic where presented credentials are valid and sufficient. Note that authentication does not determine which entities should be given access, but only verifies that an entity is who they claim to be. However, it is only after an entity is authenticated that their rights to resources can be assessed (through authorization). Hence, failure to correctly authenticate users on the Internet can leave on-line resources vulnerable to misuse.

This article considers the semantics, methods, and mechanisms for authentication on the Internet. The goals and principles of authentication are illustrated through several expository systems. The embodied trust, operation, and limitations of these systems are explored. This article is concluded with a number of axioms for the selection and use of authentication systems on the Internet.

AUTHENTICATION

An authentication process establishes the identity of some entity under scrutiny. For example, a traveler authenticates herself to a border guard by presenting a passport. Possession of the passport and resemblance to the attached photograph is deemed sufficient proof that the traveler is the identified person. The act of validating the passport (by checking a database of known passport serial numbers) and assessing the resemblance of the traveler is a form of authentication.

On the Internet, authentication is somewhat more complex; network entities do not typically have physical access to the parties they are authenticating. Malicious users or programs may attempt to obtain sensitive information, disrupt service, or forge data by impersonating valid entities. Distinguishing these malicious parties from valid entities is the role of authentication, and is essential to network security.

Successful authentication does not imply that the authenticated entity is given access. An authorization process uses authentication, possibly with other information, to make decisions about whom to give access. For example, not all authenticated travelers will be permitted to enter the country. Other factors, such as the existence of Visas, past criminal record, and political climate will determine which travelers are allowed to enter the country.

While the preceding discussion has focused on *entity authentication*, it is important to note that other forms of authentication exist. In particular, *message authentication* is the process by which a particular message is associated with some sending entity. This article restricts itself to entity authentication, deferring discussion of forms to other entries in this encyclopedia.

Meet Alice and Bob

Authentication is often illustrated through the introduction of two protagonists, Alice and Bob. In these descriptions, Alice attempts to authenticate herself to Bob. Note that in

practice, Alice and Bob may not be users, but computers. For example, a computer must authenticate itself to a file-server prior to being given access to its contents. Independent of whether Alice is a computer or person, she must present evidence of her identity. Bob evaluates this evidence, commonly referred to as a credential. Alice is deemed authentic (authenticated) by Bob if the evidence is consistent with information associated with her claimed identity. The form of Alice's credential determines the strength and semantics of authentication.

The most widely used authentication credential is a password. To illustrate, UNIX passwords configured by system administrators reside in the `/etc/passwd` file. During the login process, Alice (a UNIX user) types her password into the host console. Bob (the authenticating UNIX operating system) compares the input to the known password. Bob assumes that Alice is the only entity in possession of the password. Hence, Bob deems Alice authentic because she is the only one who could present the password (credential).

Note that Bob's assertion that Alice is the only entity who could have supplied the password is not strictly accurate. Passwords are subject to *guessing* attacks. Such an attack continually retries different passwords until the authentication is successful (the correct password is guessed). Many systems combat this problem by disabling authentication (of that identity) after a threshold of failed authentication attempts. The more serious *dictionary* attack makes use of the UNIX password file itself. A salted non-invertible hash of the password is recorded in the password file. Hence, malicious parties cannot obtain the password directly from `/etc/passwd`. However, a malicious party who obtains the password file can mount a dictionary attack by comparing hashed, salted password guesses against the password file's contents. Such an attack bypasses the authentication service, and hence is difficult to combat. Recent systems have sought to mitigate attacks on the password file by placing the password hash values in a highly restricted *shadow password* file.

Passwords are subject to more fundamental attacks. In one such attack, the adversary simply obtains the password from Alice directly. This can occur where Alice "shares" her password with others, or where she records it in some obvious place (on her PDA). Such attacks illustrate an axiom of security: a system is only as secure as the protection afforded to its secrets. In the case of authentication, failure to adequately protect credentials from misuse can result in the compromise of the system.

The definition of the identity has historically been controversial. This is largely because authentication does not truly identify physical entities, but associates some secret or (presumably) unforgeable information with a virtual identity. Hence, for the purposes of authentication, any entity in possession of Alice's password is Alice. The strength of authentication is determined by the difficulty with which a malicious party can circumvent the authentication process and incorrectly assume an identity. In our above example, the strength of the authentication process is largely determined by the difficulty of guessing Alice's password. Note that other factors, such as whether Alice chooses a poor password or writes it on the front of the monitor will determine the effectiveness of authentication. The lesson here is that authentication is as strong as the weakest link; failure to protect the password either by Alice or the host limits the effectiveness of the solution.

Authentication on the Internet is often more complex than suggested by the previous example. Often, Alice and Bob are not physically co-located. Hence, both parties will wish to authenticate each other. In our above example, Alice will wish to ensure that she is communicating with Bob. However, no formal process is needed; because Alice is sitting at the terminal, she assumes that Bob (the host) is authentic.

On the Internet, it is not always reasonable to assume that Alice and Bob have (or can) established some relationship prior to communication. For example, consider the case where Alice is purchasing goods on the Internet. Alice goes to Bob's web server, identifies the goods she wishes to purchase, provides her credit card information, and submits the transaction. Alice, being a cautious customer, wants to ensure that this information is only being given to Bob's web server (i.e., authenticate the web server). In general, however, requiring Alice to establish a direct relationship with each vendor from whom she may purchase goods is not feasible (e.g., not feasible to establish passwords for each website out-of-band).

Enter Trent, the trusted third party. Logically, Alice appeals to Trent for authentication information relating to Bob. Trent is trusted by Alice to assert Bob's authenticity. Therefore, Bob need only establish a relationship with Trent to begin communicating with Alice. Because the number of widely used trusted third parties is small (on the order of tens), and every website establishes a relationship with at least one of them, Alice can authenticate virtually every vendor on the Internet.

CREDENTIALS

Authentication is performed by the evaluation of credentials supplied by the user (i.e. Alice). Such credentials can take the form of something you know (e.g., password), something you have (e.g., smartcard), or something you are (e.g., fingerprint). The credential type is specific to the authentication service, and reflects some direct or indirect relationship between the user and the authentication service.

Credentials often take the form of shared secret knowledge. Users authenticate themselves by proving knowledge of the secret. In the UNIX example above, knowledge of the password is deemed sufficient evidence to prove user identity. In general, such secrets need not be statically defined passwords. For example, users in one-time password authentication systems do not present knowledge of secret text, but identify a numeric value valid only for a single authentication. Users need not present the secret directly, but demonstrate knowledge of it (e.g., by presenting evidence that could only be derived from it).

Secrets are often long random numbers, and thus can not be easily remembered by users. For example, a typical RSA private-key is a 1024-digit binary number. Requiring a user to remember this number is at the very least, unreasonable. Such information is frequently stored in a file on a user's host computer, a PDA, or other non-volatile storage. The private key is used during authentication by accessing the appropriate file. However, private keys can be considered "secret knowledge" because the user presents evidence external to the authentication system (e.g., from the file-system).

Credentials may also be physical objects. For example, a smartcard may be required to gain access to a host. Authenticity in these systems is inferred from possession, rather than knowledge. Note that there is often a subtle difference between the knowledge and possession based credentials. For example, it is often the case that a user-specific private key is stored on an authenticating smartcard. In this case, however, the user has no ability to view or modify the private key. The user can only be authenticated via the smartcard issued to the user. Hence, for the purposes of authentication, the smartcard is identity; no amount of effort can modify the identity encoded in the smartcard.[1]

Biometric devices measure physical characteristics of the human body. An individual is deemed authentic if the measured aspect matches previously recorded data. The accuracy of matching determines the quality of authentication. Contemporary biometric devices include fingerprint, retina, or iris scanners, and face recognition software. However, biometric devices are primarily useful only where the scanning device is trusted (i.e., under control of the authentication service). While biometric authentication has seen limited use in the Internet, it is increasingly used to support authentication associated with physical security (i.e., governing clean-room access).

WEB AUTHENTICATION

One of the most prevalent uses of the Internet is web browsing. Users access the web via specialized protocols that communicate HTML and XML requests and content. The requesting user's web browser renders received content. However, it is often necessary to restrict access to web content. Moreover, the interactions between the user and a web server are often required to be private. One aspect of securing content is the use of authentication to establish the true or virtual identity of clients and web servers.

Password Based Web Access

Web servers initially adopted well-known technologies for user authentication. Foremost among these was the use of passwords. To illustrate the use of passwords on the web, the following describes the configuration and use of *basic authentication* in the Apache web server (Apache, 2002). Note that the use of basic authentication in other web servers is largely similar.

Access to content protected by basic authentication in the Apache web server is indirectly governed by the password file. Web-site administrators create the password file (whose location is defined by the web-site administrator) by entering user and password information using the `htpasswd` utility. It is assumed that the passwords are given to the users using an *out of band channel* (e.g., via email, phone).

In addition to specifying passwords, the web server must identify the subset of web content to be password protected (e.g., set of protected URLs). This is commonly performed by creating a `.htaccess` file in the directory to be protected. The `.htaccess` file defines the authentication type and specifies the location of the relevant password

[1] In practice, contemporary smartcards can be modified or probed. However, because such manipulation often takes considerable effort and sophistication (e.g. use of electron microscope), such attacks are beyond the vast majority of attackers.

file. For example, located in the content root directory, the following `.htaccess` file restricts access to those users who are authenticated via password.

```
AuthName "Restricted Area"
AuthType Basic
AuthUserFile /var/www/webaccess
require valid-user
```

Users accessing protected content (via a browser) are presented with a password dialog (e.g., similar to the dialog depicted in **Error! Reference source not found.**). The user enters the appropriate username and password, and if correct, is given access to the web content.

Because basic authentication sends passwords over the Internet in clear-text, it is relatively simple to recover them by eavesdropping the HTTP communication. Hence, basic authentication is sufficient to protect content from casual misuse, but should not be used to protect valuable or sensitive data. However, as is commonly found on commercial websites, performing basic authentication over more secure protocols (e.g., SSL, *see below*) can mitigate or eliminate many of the negative properties of basic authentication.

Many password-protected websites store user passwords (in encrypted form) in *cookies* the first time a user is authenticated. In these cases, the browser automatically submits the cookie to the website with each request. This approached eliminates the need for the user to be authenticated every time she visits the website. However, this convenience has a price. In most single user operating systems, any entity using the same host will be logged in as the user. Moreover, the cookies can be easily captured and replayed back to the website (Fu et al 2000).

Digest Authentication uses *challenges* to mitigate the limitations of password-based authentication (Franks et al., 1999). Challenges allow authenticating parties to prove knowledge of secrets without exposing (transmitting) them. In digest authentication, Bob sends a random number (*nonce*) to Alice. Alice responds with a hash of the random number and her password. Bob uses Alice password (which only he and Alice know) to compute the correct response, and compares it to the one received from Alice. Alice is deemed authentic if the computed and received responses match (because only Alice could have generated the response). Because the hash, rather than the secret, is sent, no adversary can obtain Alice's password from the response.

A number of other general-purpose services have been developed to support password maintenance. For example, RADIUS, DIAMETER, and LDAP password services have been widely deployed on the Internet. Web servers or hosts subscribing to these services defer all password maintenance and validation to a centralized service. While each system may use different services and protocols, users see interfaces similar to those presented by basic authentication (e.g., user login above). However, passwords are maintained and validated by a centralized service, rather than by the web server.

Single Sign-on

Basic authentication has become the predominant method of performing authentication on the web. Users often register a username and password with each retailer or service provider with which they do business. Hence, users are often faced with the difficult and error prone task of maintaining a long list of usernames and passwords. In practice, users avoid this maintenance headache by using the same passwords on all websites. However, this allows adversaries who gain access to the user information on one site to impersonate the user on many others.

A *single sign-on* system (SSO) defers user authentication to a single, universal authentication service. Users authenticate themselves to the SSO once per session. Subsequently, each service requiring user authentication is redirected to a SSO server that vouches for the user. Hence, the user is required to maintain only a single authentication credential (e.g., SSO password). Note that the services themselves do not possess user credentials (e.g., passwords), but simply trust the SSO to state which users are authentic.

While single sign-on services have been used for many years (e.g., see *Kerberos* below), the lack of universal adoption and cost of integration has made their use in web applications highly undesirable. These difficulties have led to the creation of SSO services targeted specifically to authentication on the web. One of the most popular of these systems is the *Microsoft passport* service (Microsoft, 2002). Passport provides a single authentication service and repository of user information. Websites and users initially negotiate secrets during the Passport registration process (i.e., user passwords and website secret keys). In all cases, these secrets are known only to the passport servers and the registering entity.

Passport authentication proceeds as follows. Users requesting a protected webpage (i.e., page that requires authentication) are redirected to a passport server. The user is authenticated via a passport supplied login screen. If successful, the user is redirected back to the original website with an authentication *cookie* specific to that site. The cookie contains user information and site specific information encrypted with a secret key known only to the site and the passport server. The website decrypts and validates the received cookie contents. If successful, the user is deemed authentic and the session proceeds. Subsequent user authentication (with other sites) proceeds similarly, save that the login step is avoided. Successful completion of the initial login is noted in a session cookie stored at the user browser and presented to the passport server with later authentication requests.

While SSO systems solve many of the problems of authentication on the web, they are not a panacea. By definition, SSO systems introduce a single point of trust for all users in the system. Hence, ensuring that the SSO is not poorly implemented, poorly administered, or malicious is essential to its safe use. For example, passport has been shown to have several crucial flaws (Kormann and Rubin, 2000). Note that while existing web-oriented SSO systems may be extended to support mutual authentication, the vast majority have yet to do so.

Certificates

While passwords are appropriate for restricting access to web content, they are not appropriate for more general Internet authentication needs. Consider the website for an on-line bookstore *examplebooks.com*. Users wishing to purchase books from this site

must be able to determine that the website is authentic. If not authenticated, a malicious party may impersonate *examplebooks.com* and fool the user into exposing his credit card information.

Note that most web-enabled commercial transactions do not authenticate the user directly. The use of credit card information is deemed sufficient evidence of the user's identity. However, such evidence is typically evaluated through the credit card issuer service (e.g., checking that the credit card is valid and has not exceeded its spending limit) before the purchased goods are provided to the buyer.

The dominant technology used for Internet website authentication is public key certificates. Certificates provide a convenient and scalable mechanism for authentication in large distributed environments (such as the Internet). Note that certificates are used to enable authentication of a vast array of other non-web services. For example, certificates are often used to authenticate electronic mail messages (see PGP below).

Certificates are used to document an association between an identity and a cryptographic key. Keys in public key cryptography are generated in pairs; a public and private key (Diffie and Hellman, 1976). As the name would suggest, the public key is distributed freely, and the private key is kept secret. To simplify, any data signed (using a digital signature algorithm) by the private key can be validated using the public key. A valid digital signature can be mapped to exactly one private key. Therefore, any valid signature can only be generated by some entity in possession of the private key.

Certificates are issued by *Certification Authorities* (CA). The CA issues a certificate by signing an identity (e.g., domain name of the website), validity dates, and a web site's public key. The certificate is then freely distributed. A user validates a received certificate by checking the CA's digital signature. Note that most browsers are installed with a collection of CA certificates that are invariantly trusted (i.e. do not need to be validated). For example, many web sites publish certificates issued by the Verisign CA (Verisign, 2002) whose certificate is installed with most browsers. In its most general form, a system used to distribute and validate certificates is called a *public key infrastructure*.

SSL

Introduced by Netscape in 1994, the SSL protocol uses certificates to authenticate web content. In addition to authenticating users and websites, the SSL protocol negotiates an ephemeral secret key. This key is subsequently used to protect the integrity and confidentiality of all messages (e.g., by encrypting the messages sent between the web server and the client). SSL continues to evolve. For example, the standardized and widely deployed TLS (Transport Layer Security) protocol is directly derived from SSL version 3.0.

The use of SSL is signaled to the browser and website through the `https` URL protocol identifier. For example, Alice enters the following URL to access a website of interest:

`https://www.example.com/`

In response to this request, Alice's browser will initiate a SSL handshake protocol. If the website is correctly authenticated via SSL, the browser will retrieve and render website

content in a manner similar to HTTP. Authentication is achieved in SSL by validating statements signed by private keys associated with the authenticated party's public key certificate.

Figure 2 depicts the operation of the SSL authentication and key agreement process. The *SSL handshake protocol* authenticates one or both parties, negotiates the cipher-suite policy for subsequent communication (e.g., selecting cryptographic algorithms and parameters), and establishes a *master secret*. All messages occurring after the initial handshake are protected using cryptographic keys derived from the master secret.

The handshake protocol begins with both Alice (the end-user browser) and Bob (the web server) identifying a cipher suite policy and session identifying information. In the second phase, Alice and Bob exchange certificates. Note that policy will determine which entities require authentication: as dictated by policy, Alice and/or Bob will request an authenticating certificate. The certificate is validated upon reception (e.g., issuance signature checked against CAs whose certificate is installed with the browser). Note that in almost all cases, Bob will be authenticated, but Alice will not. In these cases, Bob typically authenticates Alice using some external means (only when it becomes necessary). For example, online shopping websites will not authenticate Alice until she expresses a desire to purchase goods, and her credit card number is used to validate her identity at the point of purchase.

Interleaved with the certificate requests and responses is the server and client key exchange. This process authenticates each side by signing information used to negotiate a session key. The signature is generated using the private key associated with the certificate of the party to be authenticated. A valid signature is deemed sufficient evidence because only an entity in possession of the private key could have generated it. Hence, signed data can be accepted as proof of authenticity. The session key is derived from the signed data, and the protocol completes with Alice and Bob sending *finished* messages.

HOST AUTHENTICATION

Most computers on the Internet provide some form of *remote access*. Remote access allows users or programs to access resources on a given computer from anywhere on the Internet. This access enables a promise of the Internet; independence from physical location. However, remote access has often been the source of many security vulnerabilities. Hence, protecting these computers from unauthorized use is essential. The means by which host authentication is performed in large part determines the degree to which an enterprise or user is protected from malicious parties lurking in the dark corners of the Internet. This section reviews the design and use of the predominant methods providing host authentication.

Remote Login

Embodying the small, isolated UNIX networks of old, *remote login* utilities allow administrators to identify the set of hosts and users that are deemed ``trusted''. Trusted hosts are authenticated by source IP address, hostname, and/or username only. Hence, trusted users and hosts need not provide a username or password.

The `rlogin` and `rsh` programs are used to access hosts. Configured by local administrators, the `/etc/hosts.equiv` file enumerates hosts/users that are trusted. Similarly, the `.rhosts` file contained in each users home directory identifies the set of hosts trusted by an individual user. When a user connects to a remote host with a remote login utility, the remote login server (running on the accessed host) scans the `hosts.equiv` configuration file for the address and username of the connecting host. If found, the user is deemed authentic and allowed access. If not, the `.rhosts` file of the accessing user (identified in the connection request) is scanned, and access is granted where the source address and username is matched.

The remote access utilities do not provide strong authentication. Malicious parties may trivially forge IP addresses, DNS records, and usernames (called *spoofing*). While recent attempts have been made to address the security limitations of the IP protocol stack (e.g. IPsec, DNSsec), this information is widely accepted as untrustworthy. Remote access tools trade-off security for ease of access. In practice, these tools often weaken the security of network environments by providing a vulnerable authentication mechanism. Hence, the use of such tools in any environment connected to the Internet is considered extremely dangerous.

SSH

The early standards for remote access, `telnet` and `ftp`, authenticated users by UNIX password. While the means of authentication were similar to terminal login, their use on an open network introduces new vulnerabilities. Primarily, these utilities are vulnerable to password sniffing. Such attacks passively listen on the network for communication between the host and the remote user.[2] Because passwords are sent in the clear (unencrypted), user-specific authentication information could be recovered. For this reason, the use of these utilities as a primary means of user access has largely been abandoned.[3]

The secure shell (SSH) (Ylonen, 1996) combats the limitations of standard tools by performing cryptographically supported host and/or user authentication. Similar to SSL, a byproduct of the authentication is a cryptographic key used to obscure and protect communication between the user and remote host. SSH is not vulnerable to sniffing attacks, and has been widely adopted as a replacement for the standard remote access tools.

SSH uses public key cryptography for authentication. Upon installation, each server host (host allowing remote access via SSH) generates a public key pair. The public key is manually stored at each initiating host (host from which a user will remotely connect). Note that unlike SSL, SSH uses public keys directly, rather than issued certificates.

---

[2] The physical media of over which many local network communication occurs is Ethernet. Because Ethernet is a broadcast technology, all hosts on the local network (subnet) receive every bit of transmitted data. Obviously, this approach simplifies communication eavesdropping. While eavesdropping may be more difficult over other network media (e.g. switched networks), it is by no means impossible.

[3] `ftp` is frequently used on the web to transfer files. When used in this context, `ftp` generally operates in *anonymous* mode. `ftp` performs no authentication in this mode, and the users are often restricted to file retrieval only.

Hence, SSH authentication relies on the due-diligence of host administrators to maintain the correct set of host keys.

SSH initiates a session in two phases. In the first phase, the server host is authenticated. The initiating host initiates the SSH session by requesting remote access. To simplify, the requesting host generates a random session key, encrypts it with received host public key of the server, and forwards it back to the server[4]. The server recovers session key using its host private key. Subsequent communication between the hosts is protected using the session key. Because only someone in possession of the host private key could have recovered the session key, the server is deemed authentic.

The second phase of SSH session initialization authenticates the user. Dictated by the configured policy, the server will use one of the following methods to authenticate the user:

- *.rhosts file* - as described for the *remote access* utilities above, simply tests whether the accessing user identifier is present in the `.rhosts` file located in the home directory of the user.
- *.rhosts with RSA* - similar to previous, but requires that the accessing host is authenticated via a known and trusted RSA public key.
- *password authentication* - prompts the user for local system password. The strength of this approach is determined by the extent to which the user keeps the password private.
- *RSA user authentication* - authenticates via a user-specific RSA public key. Of course, this requires that the server be configured with the public key generated for each user.

Note that it is not always feasible to obtain the public key of each host that a user will access. Host keys may change frequently (as based on an administrative policy), be compromised, or accidentally deleted. Hence, where the remote host key is not known (and the configured policy allows it), SSH will simply transmit it during session initialization. The user is asked if the received key should be accepted. If accepted, the key is stored in the local environment and is subsequently used to authenticate the host.

While the automated key distribution mode does provide additional protection over conventional remote access utilities (e.g., sniffing prevention), the authentication mechanism provides few guarantees. A user accepting the public key knows little about its origin (e.g., is subject to forgery, man-in-the-middle attacks, etc.). Hence, this mode may be undesirable for some environments.

One-Time Passwords

In a very different approach to combating password sniffing, the S/Key system (Haller, 1994) limits the usefulness of recovered passwords. Passwords in the S/Key system are valid only for a single authentication. Hence, a malicious party gains nothing by recovery of a previous password (e.g., via eavesdropping of a telnet login).

[4] In reality, the server initially transmits a short-term public key in addition to the host key. The requesting host encrypts the random value response with both keys. The use of the short-term keys prevents adversaries from recovering the content of past sessions should the host key become compromised.

While on the surface one-time password approach may seem to require that the password be changed following each login, the way in which passwords are generated alleviates the need for repeated coordination between the user and remote host.

The S/Key system establishes an ordered list of passwords. Each password is used in order and only once, then discarded. While the maintenance of the password list may seem like an unreasonable burden to place on a user, the way in which the passwords are generated makes it conceptually simple. Essentially, passwords are created such that the knowledge of a past password provides no information on future passwords. However, if one knows a secret value (called a seed value), then all passwords are easily computable. Hence, while an authentic user can supply passwords as they are needed, a malicious adversary can only supply those passwords that have been previously used (and are no longer valid).

In essence, the S/Key system allows the user to prove knowledge of the password without explicitly stating it. Over time, this relatively simple approach has been found to be extremely powerful, and is used as the basis of many authentication services. For example, RSA's widely used SecurID combines a physical token with one-time password protocols to authenticate users (RSA, 2002).

Kerberos

The Kerberos system (Neuman and T. Ts'o, 1994) performs *trusted third party* authentication. In Kerberos, users, hosts and services defer authentication to a mutually trusted Key Distribution Center (KDC). All users implicitly trust the KDC to act in their best interest. Hence, this approach is appropriate for localized environments (e.g., campus, enterprise), but does not scale well to large, loosely coupled communities. Note that this is not an artifact of Kerberos system, but true of any trusted third party approach; loosely coupled communities are unlikely to universally trust a single authority.

Depicted in Figure 3, the Kerberos system performs mediated authentication between Alice and Bob through a two-phase exchange with the KDC. When logging onto the system, Alice enters her username and password.

Alice's host sends the KDC her identity. In response, the KDC sends Alice information that can only be understood by someone in possession of the password (is encrypted with a key derived from the password). Included in this information is a ticket granting ticket (TGT) used later by Alice to initiate a session with Bob. Alice is deemed authentic because she is able to recover the TGT.

At some later point, Alice wishes to perform mutual authentication with another entity, Bob. Alice informs the KDC of this desire by identifying Bob and presenting the previously obtained TGT. Alice receives a message from the KDC containing the session key and a ticket for Bob. Encrypting the message with a key known only to Alice ensures that its contents remain confidential.

Alice then presents the ticket included in the message to Bob. Note that the ticket returned to Alice is opaque; its contents are encrypted using a key derived from Bob's password. Therefore, Bob is the only entity that can retrieve the contents of the ticket. Because later communication between Alice and Bob uses the session key (given to Alice

and contained in the ticket presented to Bob), Alice is assured that Bob is authentic. Bob is assured that Alice is Authentic because Bob's ticket explicitly contains Alice's identity.

One might ask why Kerberos uses a two-phase process. Over the course of a session, Alice may frequently need to authenticate a number of entities. In Kerberos, because Alice obtains a TGT at login, later authentication can be performed automatically. Thus, the repeated authentication of users and services occurring over time does not require human intervention; Alice types in her password exactly once.

Because of its elegant design and technical maturity, the Kerberos system has been widely accepted in local environments. Historically common in UNIX environments, it has recently been introduced into other operating systems (e.g., Windows 2000, XP).

Pretty-Good-Privacy

As indicated by the previous discussion of trusted third parties, it is often true that two parties on the Internet will not have a direct means of performing authentication. For example, a programmer in Great Britain may not have any formal relationship with a student in California. Hence, no trusted third party exists to which both can defer authentication. A number of attempts have been made to address this problem by establishing a single public key infrastructure spanning the Internet. However, these structures require users directly or indirectly trust CAs whose operation they know nothing about. Such assumptions are inherently dangerous, and have been largely rejected by user communities.

The Pretty-Good-Privacy system (Zimmermann, 1994) takes advantage of informal social and organization relationships between users on the Internet. In PGP, each user creates a self-signed PGP certificate identifying a public key and identity information (e.g., e-mail address, phone number, name). Users use the key to sign the keys of those users that they trust. Additionally, they obtain signatures from those users who trust them. The PGP signing process is not defined by PGP. Users commonly will exchange signatures with friends and colleagues.

The keys and signatures defined for a set of users defines a *web of trust*. Upon reception of a key from a previously unknown source, an entity will make a judgment of whether to accept the certificate based on the presence of signatures by known entities. A certificate will likely be accepted if a signature generated by a trusted party (with known and acceptable signing practices) is present. Such assessment can span multiple certificates, where signatures create trusted linkage between acceptable certificates. However, because trust is not frequently transitive, less trust is associated with long chains. PGP certificates are primarily used for electronic mail, but have been extended to support a wide range of data exchange systems (e.g., Internet newsgroups).

The PGP approach and other technologies are used as the basis of S/MIME standards (Dusse et al., 1998). S/MIME defines protocols, data structures, and certificate management infrastructure for authentication and confidentiality of MIME (Multipurpose Internet Mail Extensions) data. These standards are being widely adopted as a means to secure personal and enterprise email.

IPsec

IPsec (Kent and Atkinson, 1998) is emerging as a important service for providing security on the Internet. IPsec is not just an authentication service, but provides a complete set of protocols and tools for securing IP based communication. The IPsec suite of protocols provides host-to-host security within the operating system implementation of the IP protocol stack. This has the advantage of being transparent to applications running on the hosts. A disadvantage of IPsec is that it does not differentiate between users on the host. Hence, while communication passing between the hosts is secure (as determined by policy), little can be ascertained as to the true identity of users on those hosts.

The central goal of the IPsec was the construction a general-purpose security infrastructure supporting many network environments. Hence, IPsec supports the use of an array of authentication mechanisms. IPsec authentication can be performed manually or automatically. Manually authenticated hosts share secrets distributed via administrators (i.e., configured manually at each host). Identity is inferred from knowledge of the secret. Session keys are directly or indirectly derived from the configured secret.

The Internet Security Association Key Management Protocol (ISAKMP) defines an architecture for automatic authentication and key management used to support the IPsec suite of protocols. Built on ISAKMP, the Internet Key Exchange protocol (IKE) implements several protocols for authentication and session key negotiation. In these protocols, IKE negotiates a shared secret and policy between authenticated end-points of an IPsec connection. The resulting IPsec Security Association (SA) records the result of IKE negotiation, and is used to drive later communication between the end-points.

The specifics of how authentication information is conveyed to a host are a matter of policy and implementation. However, in all implementations, each host must identify the keys or certificates to be used by IKE authentication. For example, Windows XP provides dialogs used to enter preshared keys. These keys are stored in the Windows registry, and are later used by IKE for authentication and session key negotiation. Note that how the host stores secrets is of paramount importance. As with any security solution, users should carefully read all documentation and related security bulletins when using such interfaces.

CONCLUSION

The preceding sections described but a small fraction of vast array of available authentication services. Given the huge number of alternatives, one might ask, ``which one of these systems is right for my environment?'' The following presents several guidelines for the integration of an authentication service with applications and environments.

- *Don't try to build a custom authentication service*. Designing and coding an authentication service is inherently difficult. This fact has been repeatedly demonstrated on the Internet; bugs and design flaws are occasionally found in widely deployed systems, and several custom authentication services have been broken in a matter of hours. It is highly likely that there exists an authentication service that is appropriate for a

given environment. For all of these reasons, one should use services that have been time-tested.

- *Understand who is trusted by whom*. Any authentication system should accurately reflect the trust held by all parties. For example, a system that authenticates students in a campus environment may take advantage of local authorities. In practice, such authorities are unlikely to be trusted by arbitrary end-points in the Internet. Failure to match the trust existing in the physical world has ultimately led to the failure of many services.
- *Evaluate the value of the resources being protected and the strength of the surrounding security infrastructure*. Authentication is only useful when used to protect access to a resource of some value. Hence, the authentication service should accurately reflect the value of the resources being protected. Moreover, the strength of the surrounding security infrastructure should be matched by authentication service. One wants to avoid ``putting a steel door in a straw house''. Conversely, a weak or flawed authentication service can be used to circumvent the protection afforded by the surrounding security infrastructure.
- *Understand who or what is being identified*. Identity can mean many things to many people. Any authentication service should model identity as is appropriate for the target domain. For many applications, it is often not necessary to map a user to a physical person or computer, but treat them as distinct, but largely anonymous entities. Such approaches are likely to simplify authentication, and provide opportunities for privacy protection.
- *Establish credentials securely*. Credential establishment is often the weakest point of a security infrastructure. For example, many web registration services establish passwords through unprotected forms (i.e., via HTTP). Malicious parties can (and have) trivially sniff such passwords, and impersonate valid users. Hence, these sites are vulnerable even if every other aspect of security is correctly designed and implemented. Moreover, the limitations of many credential establishment mechanisms are often subtle. One should be careful to understand the strengths, weaknesses, and applicability of any solution to the target environment.

In the end analysis, an authentication service is one aspect of a larger framework for network security. Hence, it is a necessary to consider the many factors that contribute to the design of the security infrastructure. It is only from this larger view that the requirements, models, and design of an authentication system emerge.

BIBLIOGRAPHY

(Apache 2002)
Available at:
http://httpd.apache.org/
(Date of access: May 22, 2002)

Diffie, W. and Hellman, M.E. (1976). New Directions in Cryptography.
IEEE Transactions on Information Theory, 6, 644-654.

S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. (March 1998). S/MIME Version 2 Message Specification. Internet Engineering Task Force, RFC 2311.

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L. (1999). HTTP Authentication: Basic and Digest Access Authentication. Internet Engineering Task Force, RFC 2617.

Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. "Dos and Don'ts of Client Authentication on the Web," 10th USENIX Security Symposium, 2001.

Haller, N.M. (1994). The S/Key One-Time Password System.
In Proceedings of 1994 Internet Society Symposium on Network and Distributed System Security, 151-157.

Kent S., and Atkinson, R.. Security Architecture for the Internet Protocol.
Internet Engineering Task Force, RFC 2401.

David P. Kormann and Aviel D. Rubin, *Risks of the Passport Single Signon Protocol*, Computer Networks, (July, 2000).

(Microsoft, 2002)
Available at:
http://www.passport.com/
(Date of access: May 22, 2002)

Neuman B.C., and Ts'o T., (1994). Kerberos: An Authentication Service for Computer Networks. IEEE Communications, 32(9), 33-38.

(RSA, 2002)
Available at:
http://www.rsasecurity.com/products/securid/
(Date of access: August 21, 2002)

(Verisign 2002)
Available at:
http://www.verisign.com/

(Date of access: May 22, 2002)

Ylonen T., (1996). SSH - Secure Login Connections Over the Internet. In Proceedings of 6th USENIX Security Symposium, 37-42.

Zimmermann, P. (1994). PGP User's Guide. Distributed by the Massachusetts Institute of Technology.

FIGURES

Figure 1 - Password Authentication on the Web

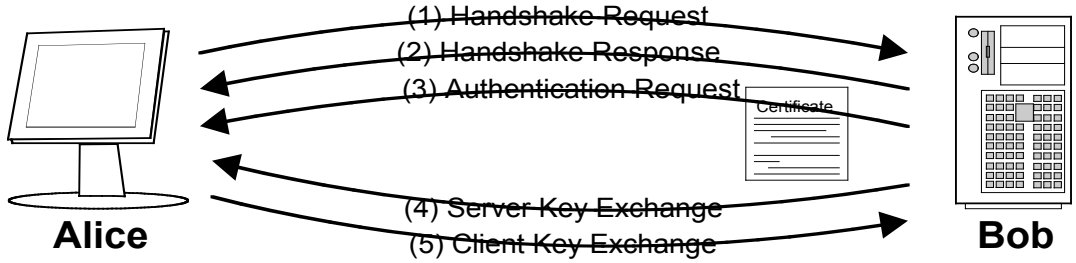

Figure 2 - The SSL Protocol: Alice (the client) and Bob (the server) exchange an initial handshake identifying the kind of authentication and configuration of the subsequent session security. As dictated by the authentication requirements identified in the handshake, Alice and Bob may exchange and authenticate certificates. The protocol completes by establishing a session-specific key used to secure (e.g., encrypt) later communication.

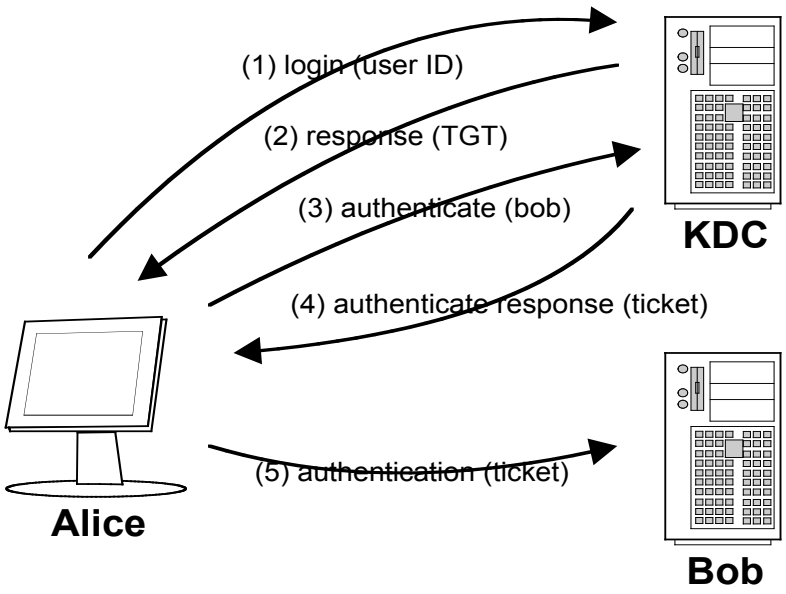

Figure 3 - Kerberos Authentication: Alice receives a ticket-granting-ticket (TGT) after successfully logging into the Kerberos Key Distribution Center (KDC). Alice performs mutual authentication with Bob by presenting a ticket obtained from the KDC to Bob. Note that Bob need not communicate with the KDC directly; the contents of the ticket serve as proof of Alice's identity.