An Authentication Framework for Web Access to Remote Hosts

Ryan P. McCormack, John E. Koontz, Judith Devaney
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

December 23, 1998

Abstract

An authentication framework is described that provides a secure means for clients to access remote computing resources via the Web. Clients authenticate themselves to a proxy Web server using a secure protocol and a digital certificate. The server constructs a fingerprint (digest) of the certificate using a secure hash function. This hash value acts as a user's identification, which is used to obtain remote login information for that user from an authentication database. Client commands can then be executed by the Web server on remote hosts using the Secure Shell protocol. The system does not require a large amount of specialized software, and creates a secure, encrypted chain between the client and any number of remote resources.

1 Introduction

Authentication is a part of everyday life in modern society, from ATM cards to driver's licenses to passports; it is essential in many instances to be able to demonstrate identity. The same is true in electronic environments like the Internet, where it is often desirable to provide access to electronic resources for a limited set of authorized users. However, in these environments, there is no recourse to physical means of identification such as photographs. One must resort to other technologies to establish identity. In the past, electronic authentication was most commonly done using login-password identification, but for most, if not all, of today's applications, this method no longer provides the needed level of security. Authentication methods exist that meet more stringent security requirements, but these are not available in all computing environments.

Another issue with the rise of the information age has been just how to gain access to all of the resources available, restricted or otherwise. The proliferation of computing platforms in the world often presents a daunting task to users. This is one major reason the World-Wide-Web has become so popular: it provides a universal interface to access a wide variety of resources. It would be useful if strong authentication technologies could be united with the Web in order to provide access to a broader spectrum of resources. This has already been done to a certain extent with the introduction of the Secure Sockets Layer (SSL) [1] protocol and digital certificates [2]. However, this type of authentication is often only a first step in the authentication process; servers may, for example, wish to gain more information about clients to provide selective access. At the present, it seems that this is typically handled on a case-by-case basis, and usually with specialized software on the server side. In addition, the resources being accessed are usually confined to the Web server host, or to other hosts accessible via the Web.

In this paper, we present a means for clients to gain access to computing resources on a group of distributed hosts by using strong authentication in conjunction with the Web. Hosts are necessarily networked, but need not be accessible via the World-Wide-Web. This unites the browser interface, recognized by most users, with an arbitrary collection of remote computing resources. The authentication framework presented uses mostly existing technologies, requires little specialized software, and can be used to deal with a wide variety of computing tasks.

2 Authentication Framework

In order to set the groundwork for the authentication framework, we first discuss the context in which authentication takes place. A model is provided that describes the authentication transaction, and the security requirements for this model are enumerated.

2.1 Model

The basic model being treated involves a three party transaction with a client, a server, and one from a collection of remote resources. The client has a valid account on the selected remote system, and wishes to perform some action on this system that requires basic login-password authentication. Once this authentication is performed, the client can perform whatever actions are allowed, given their privileges as a valid user. The primary goal of the present work is to allow the client to transparently access the remote resource using a Web browser, without requiring them to present their password or login. It is also highly desired that such a system have a low impact on all parties involved, otherwise its use would not be desirable compared to more traditional approaches. Hence, it should use existing technologies where possible, and require a minimum amount of specialized software otherwise.

The basic transaction begins when the client requests access to remote resources with their browser. The client provides authentication to the server, and the server then propagates this authentication to connect to the remote resource. This authentication process can be broken into three stages (see Fig. 1):

Stage I Client-to-Server authentication

Stage II Identity establishment and authentication translation

Stage III Server-to-Remote execution of client requests

The client provides authentication to the browser once at the outset of a session (usually by giving a password for a local certificate database). This single authentication then offers access to any one of the remote resources on which the client has privileges. This is in distinct contrast to the standard model of login-password authentication with applications such as `telnet`, `ftp`, and `rlogin`. In these systems, a login and password are normally presented for each resource accessed.

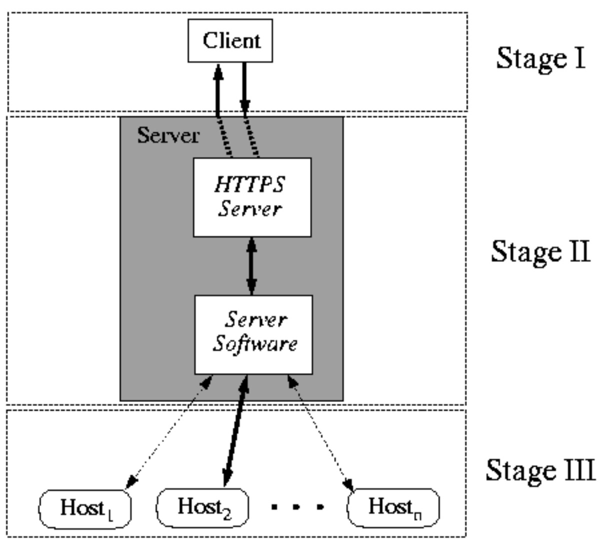

Figure 1: The three stages of authentication and remote execution

In the present scheme, the server is responsible for establishing the client's identity, translating this identity into a login name on the remote system, and then executing the client's request. A variety of implementations could be used to accomplish these tasks on the server. The simplest method is to use a collection of executable (CGI) scripts and supporting code, customized to support the applications being accessed on the remote systems. This is the approach used by WebSubmit, a Web-based interface to the high-performance computing systems at NIST.[3, 4] The method of Web-based authentication being discussed is not limited to a CGI implementation. In the remainder of the paper, however, we will assume that client requests are handled via CGI scripts, as is execution of tasks on remote systems.

2.2 Requirements

In the present context, there are two simple security policy requirements that must be met:

- Authorized users shall have access to remote hosts with all of the privileges granted to them by a normal login session.
- Unauthorized users shall not be able to access protected resources on either the server or remote host systems.

Traditional login-password authentication is often used to satisfy both of these requirements. However, this method of authentication is not completely secure in the sense that it requires the transmission of sensitive data (passwords) over an insecure network. The goal of the present security model is to provide greater security than that offered by this type of approach. In addition, it may be desirable to protect the client's data during its transactions with the remote system. The goals of the Web-based authentication framework can thus be summarized as follows:

- Construct an authentication mechanism that provides clients with Web-based access to remote resources
- Provide a level of security that exceeds that offered by traditional login-password authentication
- Create a framework that uses existing technologies where possible, and requires little or no specialized software on either the client or the remote systems.

In the remainder of the paper, we will discuss the implementation of such an authentication framework.

3 Client-to-Server Authentication

In the first stage of the authentication process, the client must authenticate itself to the server using a Web browser. This transaction is to be mediated by a Web server running on the server machine. At present, there are two standard methods for performing this type of authentication: (1) basic HTTP authentication using a login-password combination, and (2) client authentication based on public-key cryptography. Basic HTTP authentication is described in the HTTP 1.1 protocol specification.[5] When attempting to access restricted resources, the server requests a login-password combination from the client, which is encoded (not encrypted) and returned to the server. The server then compares the information presented against a database of registered users. Basic HTTP authentication is insecure since cleartext passwords are transmitted across the network, [5] and in fact may be worse than standard login-password methods (depending on how closely HTTP traffic is monitored on the server). It is subject to password sniffing and dictionary attack (repeated login attempts using a known login name with passwords taken from a carefully-chosen dictionary); it also does not provide the possibility of protecting the user's data during the transaction. Based on these two concerns, basic HTTP authentication was not deemed to be robust enough for the desired system.

Client authentication based on public-key cryptography can be implemented using a Web server that implements the Secure Sockets Layer (SSL) protocol. [1] This protocol allows for strong authentication (superior to traditional methods) and also provides data encryption over the duration of the session. It has become the *de facto* standard for secure communication on the Internet, and is in the process of being upgraded to an Internet standard (TLS - Transport Layer Security [6]). Finally, all recent versions of the two dominant Web browsers support SSL. Based on these facts, SSL-based client authentication was chosen to perform the Client-to-Server stage of the authentication process.

3.1 SSL and Digital Certificates

SSL uses a combination of public- and symmetric-key cryptography [7] to perform authentication and encryption. Public-key authentication is performed using digital certificates, and allows for the exchange of a shared secret, which is then used as an encryption key with a symmetric algorithm (e.g., DES [7]). In the present work, we require that authentication occurs in both directions: the client authenticates itself to the server and vice-versa. SSL also supports server-only authentication and anonymous sessions, although these protocols are not of interest in the present application.

Digital certificates are basically containers for public keys, and they act as a means of electronic identification (see reference [2] for a more thorough discussion of certificates). The certificate and public key are public documents that, in principle, anyone can possess. An associated private key, only possessed by the entity to whom the certificate was issued, is used as a means of binding the certificate to that entity. Users not in possession of this private key cannot use the certificate as a means of authentication. Entities can prove their possession of the private key by digitally signing known data, or by demonstrating knowledge of a secret exchanged using public-key cryptographic methods.

In practice, anyone can generate public-private key pairs and digital certificates, hence it is necessary to determine whether the holder of a certificate is to be trusted. Identity is a common means of establishing trust; restricted resources are only offered to certain individuals, and by establishing an entity's identity, one can determine whether or not to grant access. In an electronic environment, one cannot resort to visual verification of identity, using a driver's license, for instance. One possible model would be to place trust in the client, i.e., to allow the client to vouch for their own identity. A second model is to place trust in a neutral third party who will vouch for the client's identity. The former model effectively decentralizes trust, because the identity verifier (in this case, a Web server) is then required to trust each individual separately. The latter approach centralizes trust in the neutral third party. History has demonstrated that trusting clients is often ill-advised, and centralizing trust simplifies matters greatly. Hence, the trusted-third-party model is utilized with digital certificates.

The trusted third party used in the realm of digital certificates is a Certificate Authority (CA). [7] A CA can either issue certificates using public keys provided by clients, or it can generate a public-private key pair for the client and then issue the certificate along with the key pair. In either case, the client must demonstrate their identity to the CA by some trusted means. For example, the client could arrange a face-to-face meeting with the CA and present proof of identity. The CA can then issue a certificate with its digital signature that contains this client's public key, as well as information about the identity of the client. This digital signature can be verified by people who have the public key of the CA, thus establishing the chain of trust from client to CA to server. Since the validity of a client's identity hinges on the CA, it is essential to utilize a CA that is trusted. Numerous commercial CAs are currently available that will issue certificates (e.g., Thawte Consulting [8], Verisign [9]). It is also possible to install and utilize commercial or public-domain certificate server software (e.g., Netscape Certificate Server, Entrust, and SSLeay). The choice of a CA is largely a policy decision to be made based on the needs of the site requiring certification.

Once a client is in possession of a key pair and a certificate signed by a CA, there are two important processes that are part of SSL session establishment:

Certificate Validation The process of verifying that the digital signature on a certificate comes from a trusted CA and is valid. Validity of the signature ensures both that the certificate is from the trusted CA, and that the data in the certificate has not been modified.

Peer Authentication The process of verifying that the peer possesses the private key associated with the public key contained in a valid certificate. This can be accomplished through a variety of techniques, the algorithm depending on the protocol.

The process of establishing an SSL session does **not** provide the Web server with the identity of the client. It merely demonstrates that the client has a valid, signed certificate that the server trusts, and that the client has the associated private key. The brief description of the protocol in the next section describes where validation and authentication occur. User identity determination involves the extraction of certificate data using other means (see Section 4.1).

3.2 SSL Connection Protocol

A rough outline of SSL with client and server authentication follows:

1. Client and server exchange greeting messages and agree upon a protocol version and cipher suite (cryptographic algorithm). Session ID information and several other pieces of data relevant to the session are also transmitted.

2. Server transmits its certificate to client, requests the client's certificate, and indicates key exchange mechanisms if necessary. In the certificate request, the server indicates acceptable CAs.

3. Client validates server's certificate. If the certificate cannot be validated, the client is usually presented with the option to accept it.

4. Client transmits its certificate to server, along with a digital signature (using the client's private key) of messages exchanged during the handshake. Data containing the means to generate a shared secret (used to encrypt session data) is transmitted using a secure key exchange algorithm selected during the previous step of the protocol. Client generates the shared secret using the agreed upon parameters.

5. Server validates the client certificate and verifies the client's signature. If verified, the server decrypts cipher parameters using its private key, then generates the shared secret using the agreed upon parameters.

6. Client and server begin to exchange encrypted application data.

4 Identity Establishment and Authentication Translation

Once a client has been authenticated by the server (i.e., they have presented a valid certificate and a verifiable signature), the second stage of the process occurs: identity establishment and authentication translation. These processes occur on the Web server host itself, and allow the client's request for remote resources to proceed to the proper host.

4.1 Identity Establishment

As stated in Section 3.2, the SSL connection process does not yield the client's identity. Additional action must be taken to obtain the certificate data and to map that to a unique identity (a userID to be used with the authentication framework). Obtaining this userID is crucial, because it allows the server to propagate the client's request to the remote system. The userID can be derived from the certificate in a variety of ways, and it should be associated with a single client. This does not preclude a single client from having multiple certificates (and hence multiple, valid userIDs); the mapping may be many-to-one from userIDs to clients.

There is information in the certificate about the client's identity (Name, Organization, Email), but this information may not necessarily be unique. One would like to construct a userID that is based not only upon this information, but also on the public key of the client. One simple solution that presents itself is to require clients to possess specially-formatted certificates that contain information about their `userID` on the system. This does not correlate the `userID` and public-key, however, and creates logistical difficulties with issuing certificates in the required format. The entire certificate itself cannot be used, since this would be cumbersome, but there is another alternative: construct a fingerprint (message digest) unique to a given certificate. Fortunately, cryptographers and mathematicians have devised and analyzed one-way (or hash) functions that accomplish precisely this task.

Message digests are used widely in cryptography for digital signature verification and for ensuring data integrity. [7] A hash function is a many-to-one function that takes an arbitrary-length input message M and constructs a fixed-length output digest or *hash* $h = H(M)$. Properties of a *secure* hash function are as follows:

1. Given M, it is easy to compute h
2. Given h, it is computationally infeasible to compute M such that $H(M) = h$ (i.e., the function is *one-way*)
3. Given M, it is computationally infeasible to find another message M' such that $H(M) = H(M')$ (*weak collision resistance*)
4. It is computationally infeasible to find any two distinct messages M and M' such that $h = H(M) = H(M')$ (*collision resistance*)

In the present context, a unique userID is determined by constructing the hash of the client's certificate using a trusted algorithm (SHA-1 or MD5, for example). In order for the userID to be unique, one must have reasonable certainty that another client's certificate will not hash to the same value. This requirement is satisfied as long as the hash function is sufficiently collision resistant.

In order to determine the userID in a web environment, code on the server must have access to the client's certificate. This can be accomplished by directing the Web server to place the base-64 encoded version of the client's certificate in the environment when needed. Server software constructs a hash of the certificate using whatever means desired, at which point the hash (userID) can be used for authentication translation. A more detailed analysis of possible attacks against this method of identity establishment is given in Section 6.4.

4.2 Authentication Translation

Once a userID has been established for a client, an authentication database is used to translate this user's ID into login information on the remote hosts. This authentication database utilizes the userID as the key for each record. Attributes of the database should include, but are not limited to, the following: user name, user e-mail address, user status within the system (active or inactive), and the user's login names on the collection of machines that can be accessed by the system. The process of registering users with the system presents logistical and administrative problems, unless the CA and the administrator of the authentication system are identical. One simple solution is for the system software to contact the administrator when unauthorized access attempts are logged. At this point, the decision can be made as to whether the access attempt was by a valid, first-time user, or by an attacker. In the former case, the new user's data can be manually entered into the authentication database by the site administrator. Automatic registration with the system (i.e., addition to the authentication database) would defeat the entire scheme, unless other secure means can be established for this task.

When a registered client makes a request to access a remote system, the user's active status is first verified. If they are not active within the system, they are not allowed access to resources. This essentially amounts to the possibility for revocation of access privileges, in addition to those provided by the client certificate's validity period and any CA revocation lists in use. Once the user's active status is verified, the userID-remote host combination is used to index into the database, which determines the login of the user on the remote system. At this point, the request can be propagated to the remote system by the server software. The entire process of identity establishment and authentication translation is summarized in Figure 2.

5 Server-to-Remote Execution

In the present architecture, the web server host acts as a proxy for handling client requests. The web server is the agent that accomplishes remote execution, performed by running a command on the server that in turn spawns the remote command. The commands on the server and remote system run under (possibly distinct)

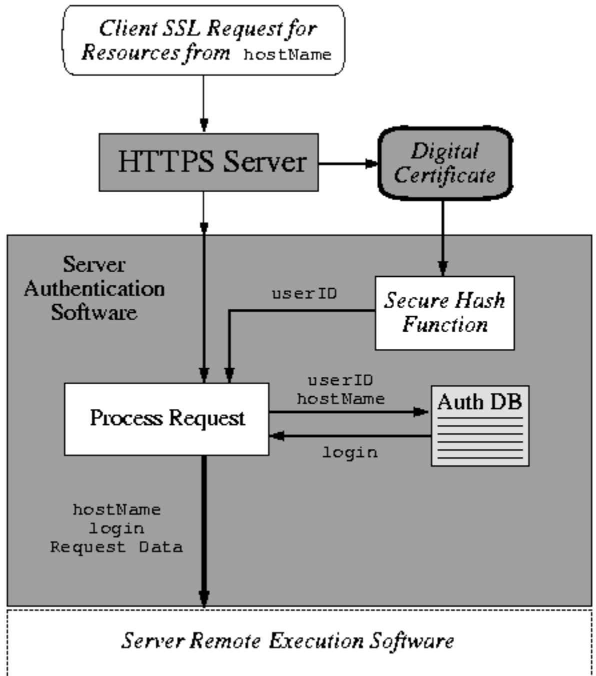

Figure 2: Identity establishment and authentication translation. The server host is contacted by the client using an SSL connection with client authentication; the client is requesting the use of resources on hostName. The client's certificate is made available to server software, which constructs a userID using a secure hash function applied to the certificate. The userID and hostName are indexed into the authentication database, which retrieves the login name for the client on hostName (if possible). The client's request is then processed, and all information is forwarded to the software to perform execution on the remote host hostName.

usernames UID_S^i and UID_R^j, respectively. Since the server is necessarily accomplishing a task for the *client* on the remote system, $\mathrm{UID}_R^j = \mathrm{UID}_R^{client}$ (the UID that corresponds to the client's login name on the remote system). In the present scheme, UID_S^i corresponds to the UID of the Web server user (`serverUser`). If the client had an account on the server, it would also be possible to have $\mathrm{UID}_S^i = \mathrm{UID}_S^{client}$ (by having the Web server invoke an SUID script). In general, however, it should be assumed that not all clients will have accounts on the server, and indeed, this is preferable from a security standpoint. The fewer accounts there are on the server host, the fewer entry points there are for unauthorized logins.

Regardless of what server-side UID is used to initiate remote command execution, there needs to be a mechanism for this execution. An explicit goal stated at the outset was that the authentication system developed should utilize existing technologies where possible. This minimizes the amount of specialized, and possibly untrusted, software running on the remote hosts. One common means for executing commands remotely on UNIX systems is via the remote shell (`rsh`) command. [10] Using appropriately configured `.rhosts` files, commands can be executed from the server host in a client's account on the remote system. However, `rsh` does not protect against the possibility of unauthorized clients masquerading as the server host. This is typically done using IP or DNS spoofing. A method of executing commands remotely that is not subject to these attacks, and that provides encryption, is the Secure Shell (SSH) protocol. [11, 12]

SSH has grown in popularity since its introduction, and is in the process of being considered for an Internet standard. The software has been ported to a wide variety of UNIX platforms; both commercial and non-commercial versions are available. SSH has several features that make it attractive in the present context:: (1) Strong authentication methods prevent identity spoofing, trojan horses, and similar means of attack, (2) Encryption and compression of data, and (3) Secure means for file transfer. These qualities precisely meet the needs of the problem being addressed, hence SSH was chosen as the means to execute commands on the remote system.

SSH uses a hybrid cryptosystem similar to SSL; a shared secret is exchanged using public-key cryptography, and then data is encrypted using a symmetric cipher based on the shared secret. The basic SSH protocol is as follows:

1. Client and server exchange greeting messages and establish an algorithm for public key exchange (Diffie-Helman in the present example).

2. Server transmits its host key to the client along with a signature verifying that it has the proper private key. Server generates a shared secret.

3. Client validates server's certificate (either using a CA or against a local database), verifies the server's signature, then generates the shared secret.

4. Client and server switch to the use of the shared secret to encrypt data. All transmissions from this point on are secure.

5. Client authenticates itself to server using public-key authentication, password authentication, or host-based authentication (present scheme).

6. Client and server exchange encrypted data within the SSH connection protocol.

In the present approach, secure host-based authentication (called `RhostsRSAAuthentication`) is used, since this allows the Web server proxy to execute commands on the remote systems *as the user*, without the need for password exchanges.

5.1 Server Configuration

All remote hosts that are to be accessed must be known to the server system in order to establish the initial encrypted communications channel. This requires placing the public host keys for these systems into `/etc/ssh_known_hosts` on the server machine. In addition, the server system needs to have `ssh` and `scp` (Secure CoPy) installed, although it is not necessary to have an SSH daemon running. The daemon is unnecessary because the server system will always be the system to initiate the SSH request.

5.2 Remote System Configuration

Remote systems must be configured to receive SSH connections and authenticate clients using the RSA host-based scheme. The remote system must have a properly configured SSH daemon (`sshd`); this daemon must be able to locate both `ssh` and `scp` on the system, and it should be set up so that it cannot revert to insecure methods of authentication. Client host public keys are stored on the remote systems to authenticate client hosts (again, typically in `/etc/ssh_known_hosts`). For RSA host-based authentication without passwords, user accounts must also be configured to allow `serverUser` to execute commands for the client (since we are using $\mathrm{UID}_S^i = \mathrm{UID}_S^{serverUser}$). This is accomplished by creating `$HOME/.shosts` (in each client account, on each remote host) with the single line

```
server.host.name serverUser
```

where `server.host.name` is the Internet host name for server host, and `serverUser` is the UID associated with the Web server. The user is necessarily providing the `serverUser` on the server host the ability to execute commands in their account on each remote system.

6 Discussion

Utilization of the authentication framework discussed here raises some questions of policy. Concern over computer security has risen noticeably in the past few years, and any new authentication system inevitably contributes to this concern. Security policies in effect on either the remote or server systems may in fact prevent use of the methods discussed here. The sections that follow discuss various issues related to policy, and an attempt is made to assess the risks associated with the use of such a system.

6.1 Certificate Authorities

Certificate authorities are a means for centralizing trust, so that the server need not trust each individual client. The server, however, must trust the CA to vouch for the identity of clients. As mentioned earlier, numerous commercial CAs exist that can issue certificates to clients. However, it may be that there is no reason to trust a commercial CA more than one would trust clients. In such a case, it will be necessary to use and maintain a CA dedicated to the system in use. The use of a CA for digital certificates raises two other concerns: the protocols used for issuing and distributing certificates to clients. Resolution of these twin concerns depends largely on the site under consideration and on the CA ultimately chosen to perform the task. Needless to say, however, the difficulty in choosing and using a CA should not be underestimated, since the CA is one very crucial link in the entire authentication process.

6.2 Firewalls

Firewalls [10] protect one network from another by filtering traffic, and their use is becoming more widespread, especially for large organizations. Many firewalls are configured to block Web server traffic (`HTTP` or `HTTPS`). In addition, many firewalls block `rsh` requests, and may consider `ssh` requests equally unreliable. For these reasons, firewall policy for the server host and the remote systems must be considered. If the server host is behind a firewall, then one must consider whether clients outside the firewall will be using the Web server to access remote hosts. If this is the case, then the firewall must pass at least HTTPS traffic to allow SSL connections to the Web server. If all clients of the system are within the firewall, then this is of no concern. In order for clients to have access to remote systems, these systems must be open to SSH traffic from the server host. Any remote system with a firewall that does not pass SSH packets from the server host will be unusable in the present scheme.

6.3 SSH

Some systems discourage the use of `.rhosts` files because of the danger these files pose to the system through identity spoofing. The use of `.shosts` files has the same result as `.rhosts` files (i.e., access without passwords), but the means through which this is achieved are totally different. SSH strong authentication essentially prevents identity spoofing. Hence, the only concern with the present approach is whether the client and remote host are comfortable with allowing the server host Web user to execute commands on behalf of the client. By accepting the server host's public SSH key, the remote system acknowledges trusting the server host. A client's trust in the server host is equivalent to the trust they place in any system administrator. The administrator of the Web server would be the only one who could act in their stead (barring root compromises of the server host). Concern regarding the trustworthiness of the server is thus in the hands of the remote system and the client; either can disable trust at any time with little effort.

The use of SSH also presents some administrative issues. On the remote system, `sshd` must be properly configured so that both `ssh` and `scp` are in the path it traverses looking for these commands. If a target system administrator moves these binaries for one reason or another, but does not recompile `sshd` to search the new location(s), then SSH to this target will not work. `sshd` should be activated during the boot cycle of the target system, otherwise normal system shutdowns may disrupt service. Finally, both the remote and server systems should pay attention to version compatibility with SSH. At the time of writing, SSH v2.0 is emerging, but not yet stable. Versions of `ssh`, `scp`, and `sshd` prior to 1.2.26 may not interoperate properly with Version 2.0, and hence the use of these earlier versions of the software should be discouraged.

6.4 Risk Assessment

As with any authentication scheme, it is important to assess the risks with the system. The present approach offers better security in general than a simple login-password scheme, such as that used by `telnet`. However, it does present a few different types of vulnerabilities that need to be addressed. For more detailed discussions about cryptographic security and general internet security, the interested reader is referred to References [7, 10].

6.4.1 Identity Establishment

Confidence in identity establishment is crucial in the present authentication scheme, hence it is worth examining in some depth. There are two basic cases that we will consider, which we will refer to as *active* attacks and *inadvertent* attacks. An active attack consists of an unauthorized user attempting to access the system; unauthorized in this context means the client does not have a valid certificate and is not registered. An inadvertent attack involves the situation where a user possesses a certificate recognized by the HTTPS server, but is not registered with the system.

In the following descriptions, active and inadvertent attacks are symbolized $A(n)$ and $I(n)$, respectively.

A1: The attacker has a registered user's certificate M: A user's certificate is of no value in and of itself, since it can only be used in an SSL connection if the connecting client also has the associated private key. If the attacker does not have this key, an attack is still possible. Assuming the hacker knows the hash algorithm used to construct userIDs, they can construct a valid `userID` $h = H(M)$. They can then attempt to construct another certificate M' such that $H(M') = h$, where the public and private key associated with this certificate are known to them. Weak collision resistance of the hash function renders this task computationally infeasible. Note that there is the additional problem that this randomly constructed certificate must also possess a valid signature. Since the CA's private key is unknown, the attacker is presented with another computationally-infeasible problem, and is thus doubly deterred.

A2: The attacker has a registered user's userID: This attack is equivalent to $A1$ (once the attacker has constructed the hash value of the user's certificate). Hence, a knowledge of the authentication database is insufficient to gain access.

A3: The attacker has no information related to registered users: In this case, the attacker must randomly generate a valid certificate that also happens to have the hash value of a registered user. This case is even more desperate than case $A1$, since testing each randomly generated certificate involves actually contacting the server to see if access is granted. In cases $A1$ and $A2$, the attacker can at least generate and test their pirate certificate behind closed doors, as it were. With repeated attempts to access the server, the attacker would hopefully be noticed due to the high volume of server hits.

I1: Two or more user's certificates yield the same hash value: Consider the case where we have two users, A and B. If A and B are both valid users, then the equivalence of their `userID` values would be noticed at registration, and a new certificate would be requested from one of the users. In the case where A is registered, but B only possesses a certificate signed by a valid CA, the situation is slightly different. In this case, the only way user B could gain access to the system was if their certificate accidentally hashed to the value held by another valid user (say A). There is a vanishing probability of this occurring if the hash function is collision-resistant.

6.4.2 Client-side Security

Clients should use browsers that support 128-bit encryption, and must ensure the security of their browser and its certificate database password. Browsers typically cache a user's certificate database password. Hence, an unattended browser at an unlocked machine could be used to connect to the system and gain access. In addition, if a user's certificate database password is not robust, it could be cracked if their account is cracked, also allowing access.

6.4.3 Attacks against SSL

SSL has been designed to be resistant to a variety of attacks. SSL Version 3.0 provides more robust security than version 2.0, and its use is recommended. The interested reader should refer to the documentation on SSLv3.0 in Reference [1] for a discussion of these attacks. In addition, it is recommended that 128-bit encryption be used for all SSL connections, thus deterring brute-force attack against a negotiated session.

6.4.4 Server Root Compromise

If the root account on the server host were to be compromised, all user accounts on remote systems would be open to access. For this reason, the security of the server machine is essential. There should be few or no user accounts on the system, and ideally, the machine should be behind a firewall. Security patches on the system should be maintained, and all unnecessary services should be disabled (e.g., `rshd` and `telnet`, dial-in access).

6.4.5 User Registration Protocol

In the case where the CA and server administrator are one and the same, user registration (i.e., addition to the authentication database) does not present any difficulties. However, the more likely case is that the CA and server administrator will be different (and possibly in different locations). In this situation, registration of users becomes a more critical issue. Ideally, the process of registration should be automated. However, this does not allow for the case where users not registered to the system are allowed access to the secure server for other reasons. User registration can be handled fairly easily in a CGI context by notifying the server administrator of unauthorized access attempts. However, if the number of users registered with the system grows large, this may be a problematic method for dealing with registration. In any case, circumventing or breaking registration protocols is a potential issue, and administrators should think through this process carefully.

6.4.6 Web Server Misconfiguration

If the Web server is misconfigured on the server host, or if the configuration is intentionally corrupted, this could alter the way identity establishment and authentication translation are performed. This would only be a problem if it occurred in conjunction with either a root compromise or with unauthorized access to another account. Otherwise, Web server misconfiguration would probably just lead to inoperable software.

6.4.7 SSHD Misconfiguration

If `sshd` is misconfigured to allow `.rhosts` authentication, the remote systems could be subject to identity spoofing through this channel. It is highly recommended to configure `sshd` so that this is not allowed (in fact, this is the default configuration).

7 Conclusion

An authentication framework has been presented that allows clients to connect to remote hosts using strong authentication methods in conjunction with the Web. The level of security provided is superior to that provided by traditional login-password approaches, and requires a minimum of specialized software on either the client or remote systems. The method presented provides the possibility for access to remote resources that might not otherwise be accessible via the interface offered by the Web.

8 Acknowledgements

The authors would like to acknowledge James Dray of the NIST security division for useful discussions regarding cryptography. John Wack was also particularly helpful in issues related to certificate authorities

and digital certificates in general.

References

[1] Netscape SSL Overview, http://home.netscape.com/eng/ssl3/ssl-toc.html

[2] R. Housley, W. Ford, W. Polk, D. Solo, PKIX Working Group Internet Draft (work in progress), http://search.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt

[3] Robert R. Lipman, Judith E. Devaney, "WebSubmit - Running Supercomputer Applications via the Web", *Proceedings of SuperComputing 96*, November 1996, Pittsburgh, PA (http://www.supercomp.org/sc96/SC96PROC/POSTER.HTM)

[4] R. McCormack, J. Koontz, J. Devaney, "WebSubmit: Web-Based Applications with Tcl", NISTIR 6165, June 1998, National Institute of Standards and Technology.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, RFC 2068, http://www.w3.org/Protocols/rfc2068/rfc2068.

[6] T. Dierks and C. Allen, Transport Layer Security Internet Draft (work in progress), http://search.ietf.org/internet-drafts/draft-ietf-tls-protocol-06.txt

[7] Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons (New York, 1996)

[8] Thawte Certification, http://www.thawte.com

[9] Verisign, http://www.verisign.com

[10] S. Garfinkel and G. Spafford, *Practical UNIX & Internet Security*, 2nd Edition, O'Reilly & Assoc. (Sebastopol, 1996)

[11] Tatu Ylönen, SSH Web site (http://www.ssh.fi)

[12] T. Ylönen, T. Kivinen, M. Saarinen, T. Rinne, S. Lehtinen, SSH Internet Draft (work in progress), http://search.ietf.org/internet-drafts/draft-secsh-architecture-02.txt