134

Understanding SSL

Chris Hare

Secure Socket Layer (SSL) is a common term in the language of the network. Users, administrators, and security professionals alike have come to learn the benefits of SSL. However, like so many technology elements, most do not understand how it works. This chapter examines what SSL is, how it works, and the role of certificates.

134.1 What Is SSL?

SSL is a method of authenticating both ends of a communication session and providing encryption services to prevent unauthorized access or modification of the data while in transit between the two endpoints. SSL is most commonly associated with protecting the data transferred in a Web browser session, although SSL is not limited to just a Web browser.

SSL is widely used in financial, healthcare, and electronic commerce applications. With the advent of SSL, users can now access banking records, make payments, and transfer funds through a financial

AU7495—Chapter134—27/1/2007—16:38—IRSHATH—14794—XML MODEL CRC12a – pp. 1777–1790.

institution's Web sites. Likewise, users can access healthcare information and even make online purchases from a favorite provider. All of this is possible without SSL; however, with the authentication and encryption capabilities, purchasers can provide their payment information immediately.

Aside from protecting Web-based transactions and other protocols, SSL is also being used to establish virtual private network (VPN) connections to a remote network.

Many network protocols in use today offer little or no protection of the data, allowing information to be transferred "in the clear." Consequently, confidentiality and integrity of the data processed in the protocol is a major concern for users and security professionals. Without additional protection, data protection is totally reliant upon the underlying network design, which itself is prone to problems.

The phenomenal growth of the Internet and its use for E-commerce, information sharing, government, and banking indicates more and more confidential information is being transferred over the Internet than ever before. SSL addresses the confidentiality issue by encrypting the data transmission between the client and server. Using encryption prevents eavesdropping of the communication. Additionally, the server is always authenticated to the client and the client may optionally authenticate to the server.

The intent of the SSL protocol was to provide higher-level protocols, such as Telnet, FTP, and HTTP, increased protection in the data stream. The protection is afforded by encapsulating the higher-level protocol in the SSL session. When establishing the connection between the client and the server, the SSL layer negotiates the encryption algorithm and session key, in addition to authenticating the server. The server authentication is performed before any data is transmitted, thereby maintaining the privacy of the session.

Developed by Netscape Communications Corporation, SSL was first proposed as an Internet Request for Comments Draft in 1994. Although never accepted as an Internet Standard by the IETF, SSL has been implemented in many commercial applications, and several open source implementations are available today.

134.2 Server Certificates

Enabling SSL requires that the application server be capable of accepting an SSL request and the existence of a server certificate. Without the server certificate, SSL is not available, even if the server is configured to offer it. The server certificate contains both public and private key components. The public certificate is provided to the client during the SSL handshake and the private component is kept on the server to verify requests and information encrypted with the server's public certificate.

The process of generating an SSL certificate is beyond the scope of the discussion. However, SSL certificates are available from a variety of certificate providers as well as OpenSSL implementations.

134.3 The SSL Handshake

There are two major phases in the SSL handshake. The first establishes the connection and authenticates the server, and the second authenticates the client. During phase 1, the client initiates the connection with the SSL server by sending a CLIENT-HELLO message.

134.3.1 The CLIENT-HELLO Message

The CLIENT-HELLO message contains a challenge from the client and the client's cipher specifications. If the client attempts to establish a connection with the SSL server using any message other than CLIENT-HELLO, it must be considered an error by the server, which in turn refuses the SSL connection request.

Within the CLIENT-HELLO message, the client specifies the following information:

- The client's SSL version
- The available cipher specifications

- A session ID if one is present
- A challenge, used for authentication

The session ID is a unique identifier indicating that the client has previously communicated with the server. If the session ID is still in the client's and the server's cache, there is no need to generate a new master key, because both ends still have a session ID from a previous connection. If the session ID is not found, then a new master key is required.

Once the client has sent the CLIENT-HELLO message to the server, the client suspends while awaiting the corresponding SERVER-HELLO message.

134.3.2 The SERVER-HELLO Message

When the server receives the CLIENT-HELLO message, it examines the provided data before responding. The server examines the parameters in the client's request, specifically to verify that it will support one of the ciphers and the client's SSL version. If the server cannot, it responds with an ERROR message to the client.

If the server can support the client's SSL version and one or more of the provided ciphers, it responds with a SERVER-HELLO message. The response includes the following information:

- The server's signed certificate
- A list of bulk ciphers and specifications
- A connection ID
- A response for the supplied SESSION ID if provided by the server

The server's signed certificate contains the server's public key, which will be used later during the connection phase if the client generates a new master key. The server provides:

- The bulk ciphers and specifications so both ends of the connection can agree upon the cipher to use in the communication
- The connection ID, which is a randomly generated value used by the client and server for a single connection

The server uses the provided SESSION ID to see if the session ID is found in the server's cache. If the session ID is not found, the server provides its certificate, and cipher specifications back to the client. The client then determines if a new master key is needed to continue the communications.

134.3.3 The CLIENT-MASTER-KEY Message

The client determines if a new master key is required, based on the response from the server for the provided session ID. The requirement for a new master key is based on the server responding positively to the provided SESSION ID, meaning that the data is in the server's cache. If the SESSION ID is not in the server's cache, then a new master key is required.

134.4 Generating a New Master Key

If a new master key is needed, the client generates the new master key using the data provided by the server in the SERVER-HELLO message and sends the new master key back to the server using a CLIENT-MASTER-KEY message. The CLIENT-MASTER-KEY message contains the following elements:

- The selected cipher chosen from the list provided by the server
- Any elements of the master key in cleartext

- An element of the master key encrypted using the server's public key
- Any data needed to initialize the key algorithm

The client uses the public key provided in the server's certificate to encrypt the new master key. After the server has received the new master key, it decrypts it using the private key corresponding to the server certificate. The master key consists of two components, one of which is transmitted to the server in the clear, and the other that is sent encrypted. The amount of master-key data sent in the clear depends on the encryption cipher in use, as explained in the section entitled "Determining the Encryption Cipher" later in this chapter.

134.4.1 Keys and More Keys

If no new master key is required, both ends of the connection generate new session keys using the existing master key, the challenge provided by the client, and the connection ID provided by the server.

The client and server use the master key to generate the session key pairs for this session. There are a total of four session keys generated, two for each end of the communication, as shown in Exhibit 134.1.

The draft Internet Request for Comments (RFC) for SSL represents the master key as a function between the server and client portions of the communications exchange. That is to say, the keys are generated using the following method:

```
CLIENT-READ-KEY=HASH(MASTER-KEY, "0," CHALLENGE, CONNECTION-ID)
SERVER-WRITE-KEY=HASH(MASTER-KEY, "0," CHALLENGE, CONNECTION-ID)
CLIENT-WRITE-KEY=HASH(MASTER-KEY, "1," CHALLENGE, CONNECTION-ID)
SERVER-READ-KEY=HASH(MASTER-KEY, "1," CHALLENGE, CONNECTION-ID)
```

The elements of the function are:

- The HASH is the cipher-specific function used to generate the keys.
- MASTER-KEY is the master key already exchanged between the client and server.
- CHALLENGE is the challenge data provided by the client in the CLIENT-HELLO message.
- CONNECTION-ID is the connection identifier provided by the server in the SERVER-HELLO message.

The "0" and "1" tell each side what key to generate. Notice the CLIENT-READ-KEY and the SERVER-WRITE-KEY both use the same "0" identifier. If they did not, the generated keys would not be related to each other and could not be used to encrypt and decrypt the data successfully. While the server is generating session keys, the client performs the same function, eliminating the need for key exchange across an untrusted network. The available ciphers are discussed later in the chapter.

134.4.2 The SERVER-VERIFY Message

Once the master key is decrypted, the server responds with a SERVER-VERIFY message. The SERVER-VERIFY response is sent after new session keys have been generated with an existing master key, or after

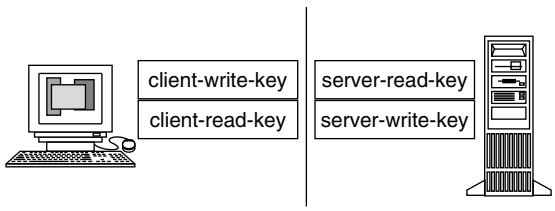

EXHIBIT 134.1 Two pairs of SSL keys are generated.

the client has sent a specific CLIENT-MASTER-KEY request. Consequently, not every SSL handshake requires an explicit CLIENT-MASTER-KEY message.

The SERVER-VERIFY message contains an encrypted version of the challenge originally sent by the client in the CLIENT-HELLO message. Only the authentic server has the private key matching the certificate, the authenticity of the server has been validated, and only the authentic server can encrypt the challenge properly using the session keys. Consequently, these two actions verify the authenticity of the server. The transaction to this point is illustrated in Exhibit 134.2.

If the client and the server cannot agree on the ciphers to use in the communication, the client returns an ERROR message to the server.

Once the keys have been generated and the server responds with the SERVER-VERIFY message, the server has been verified and phase 2 is started.

Phase 2 consists of authenticating the client, as the server is authenticated in phase 1. The server sends a message to the client requesting additional information and credentials. The client then transmits them to the server or, if it has none, responds with an ERROR response. The server can ignore the error and continue, or stop the connection, depending on how the implementation is configured.

134.4.3 The CLIENT-FINISHED and SERVER-FINISHED Messages

When the client has finished authenticating the server, it sends a CLIENT-FINISHED message with the connection ID encrypted using the client's write key (client-write-key). However, both ends of the connection must continue to listen for and acknowledge other messages until they have both sent and received a FINISHED message. Only then has the SSL handshake completed (see Exhibit 134.3).

In most cases, the SSL handshake is completed without any further effort, as rarely does the server authenticate the client. Client authentication is typically through client certificates, which are discussed later in the chapter.

134.5 Determining the Encryption Cipher

The encryption cipher is negotiated between the client and the server, based upon the cipher specifications provided in the CLIENT-HELLO and SERVER-HELLO messages. The available ciphers are:

- RC4 and MD5
- 40-bit RC4 and MD5

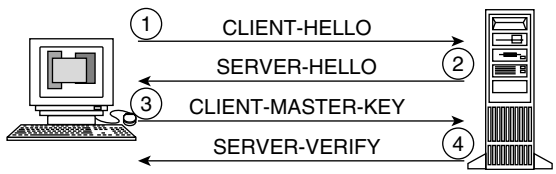

EXHIBIT 134.2 The SERVER-VERIFY message.

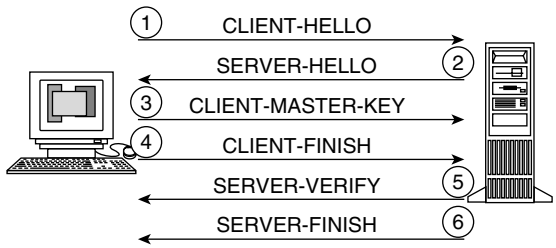

EXHIBIT 134.3 The full SSL handshake.

- RC2 with CBC and MD5
- 40-bit RC2 with CBC and MD5
- IDEA with CBC and MD5

The MD5 128-bit key is not used in the encryption. The actual encryption algorithm used in the SSL data transfer is RC2, RC4, or IDEA, with key sizes ranging from 40 to 128 bits. The actual length of the encryption key depends on the cipher negotiation. The use of cryptography and specific key lengths is often controlled by international legislation, affecting the available ciphers.

While this is not an exhaustive list and other encryption protocols may be supported, the available ciphers offer protection of the data. However, the 40-bit ciphers operate differently. When using the RC4 and RC2 ciphers, the entire session key is sent encrypted between the client and the server. However, in SSL Version 1, the 40-bit ciphers were limited to a maximum key length of 40 bits. Consequently, it is possible for the client and the server not to have a cipher they can agree upon, meaning they cannot communicate.

With SSL Version 2, the key became 128 bits regardless of implementation. However, with the EXPORT40 implementations, only 40 bits of the session key are encrypted—the other 88 bits are not.

A discussion of the encryption algorithms used is beyond the scope of this discussion; the reader is urged to review the appropriate cryptography references for information on ciphers.

134.6 Client Certificates

Unlike server certificates that are involved in phase 1 of the SSL handshake, client certificates are part of phase 2. The REQUEST-CERTIFICATE and CLIENT-CERTIFICATE messages are used during phase 2.

Client certificates must be generated or acquired and installed in the application. The process of certification acquisition and installation is outside the scope of this discussion.

134.6.1 The REQUEST-CERTIFICATE Message

The REQUEST-CERTIFICATE message is sent from the server to the client when the server has been configured to require this authentication element. The message contains:

- The desired authentication type
- A challenge

The desired authentication types are:

```
SSL_AT_MD5_WITH_RSA_ENCRYPTION
```

This message requires that the client responds with a CLIENT-CERTIFICATE message (see the following section) by constructing an MD5 message digest of the challenge and encrypting it with the client's private key. The server can then validate the authenticity when the CLIENT-CERTIFICATE message is received by performing the same MD5 digest functions, decrypting the data sent using the client's public key, and comparing it with its own MD5 digest. If the values match, the client has been authenticated.

134.6.2 The CLIENT-CERTIFICATE Message

The CLIENT-CERTIFICATE message, sent in response to a REQUEST-CERTIFICATE from the server, provides the information for the server to authenticate the client. The CLIENT-CERTIFICATE message contains the following information:

- The certificate type
- The certificate data
- The response data

However, if the client has no certificate installed, the client provides a NO-CERTIFICATE-ERROR to the server, generally meaning that the connection is refused. The certificate type used on the client side is generally an X.509 signed certificate provided by an external certificate authority.

When assembling the response to the server, the client creates a digital signature of the following elements:

- The CLIENT-READ-KEY
- The CLIENT-WRITE-KEY
- The challenge data from the REQUEST-CERTIFICATE message
- The server's signed certificate from the SERVER-HELLO message

The digital signature is encrypted with the client's private key and transmitted to the server. The server can then verify the data sent and accept the authenticity if the data is valid.

Other authentication types can be used between the client and the server and can be added by either defining a new authentication type or by changing the algorithm identifier used in the encryption engines.

134.7 Message Flow

To clarify the discussion to this point, the following examples illustrate the message flow between the client and the server—the handshake. As is evident from discussing the various messages in the protocol, there are several variations possible in establishing the connection between the client and the server.

134.7.1 Session Identifier Available

This is the simplest example of message flows in the SSL transaction. It occurs when the client and the server have the session in their cache (see Exhibit 134.4).

1. The client initiates the connection and sends the CLIENT-HELLO message, which includes the challenge, session identifier, and cipher specifications.
2. The server responds with a SERVER-HELLO message and provides the connection identifier and server hit flag.
3. The client sends the server a CLIENT-FINISH message with the connection identifier and the client-write-key. Remember that the connection identifier is encrypted with the client-write-key.
4. The server provides the original challenge from the client encrypted with the server-write-key in the SERVER-VERIFY message.

And finally, the server transmits the SERVER-FINISH message with the session identifier encrypted with the server write key.

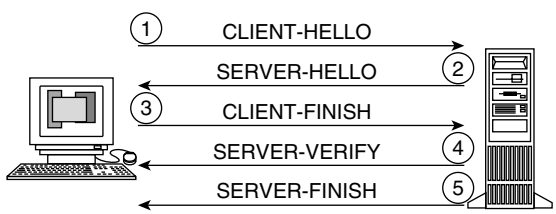

EXHIBIT 134.4 SSL session identifier available.

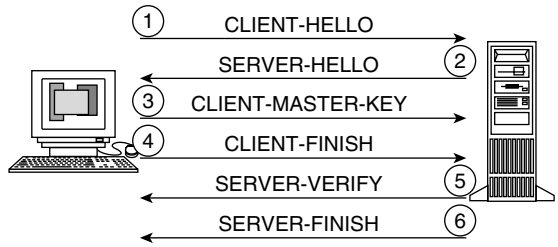

EXHIBIT 134.5 No session identifier.

134.7.2 No Session Identifier Available

This situation occurs when:

- The client has an identifier but the server does not.
- Neither the client nor the server has an identifier.

In this scenario (see Exhibit 134.5), the client connects and because there is no existing session identifier, the node must generate a new master key.

1. The client initiates the connection and sends the CLIENT-HELLO message, which includes the challenge and cipher specifications.
2. The server responds with a SERVER-HELLO message and provides the connection identifier, server certificate, and cipher specification.
3. The client selects the cipher, generates a new master key, and sends it to the server after encrypting it with the server's public key. This is the CLIENT-MASTER-KEY message.
4. The client sends the server a CLIENT-FINISH message with the connection identifier and the client-write-key. Remember that the connection identifier is encrypted with the client-write-key.
5. The server provides the original challenge from the client encrypted with the server-write-key in the SERVER-VERIFY message.

Finally, the server transmits the SERVER-FINISH message containing the new session identifier encrypted with the server-write-key.

134.7.3 The Entire Handshake Illustrated

This final example, shown in Exhibit 134.6, illustrates an SSL connection where the client must provide the new master key, new session keys are generated on both systems, and the server requests a client certificate.

1. The client initiates the connection and sends the CLIENT-HELLO message, which includes the challenge and cipher specifications.
2. The server responds with a SERVER-HELLO message and provides the connection identifier, server certificate, and cipher specification.
3. The client selects the cipher and generates a new master key, and sends it to the server after encrypting it with the server's public key. This is the CLIENT-MASTER-KEY message.
4. The client sends the server a CLIENT-FINISH message with the connection identifier and the client-write-key. Remember that the connection identifier is encrypted with the client-write-key.
5. The server provides the original challenge from the client encrypted with the server-write-key in the SERVER-VERIFY message.
6. The server sends the REQUEST-CERTIFICATE to the client, including the authentication type and challenge, encrypted with the server-write-key.
7. The client responds to the server, sending a CLIENT-CERTIFICATE message with the certificate type, the actual certificate, and the response to the challenge in the REQUEST-CERTIFICATE. All of the data is encrypted using the client-write-key.

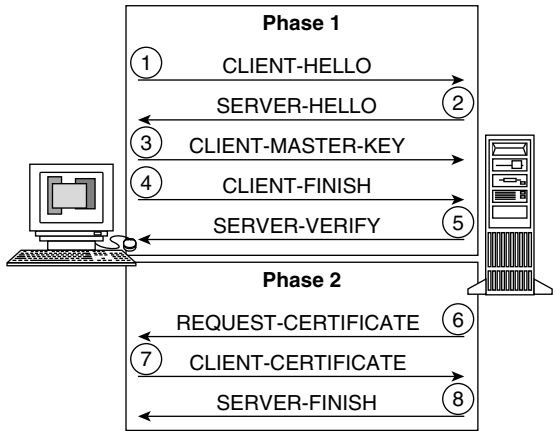

EXHIBIT 134.6 The complete SSL handshake.

Finally, the server transmits the SERVER-FINISH message containing the new session identifier encrypted with the server-write-key.

134.8 Is It All Encrypted?

The answer is no. Not all information during the handshake is actually sent encrypted, depending upon the phase of the handshake. Specifically, the following elements of the handshake are not encrypted:

- The CLIENT-HELLO message
- The SERVER-HELLO message
- The CLIENT-MASTER-KEY message
- The CLIENT-FINISHED
- SERVER-HELLO
- SERVER-FINISHED

Despite the messages that are not encrypted, sufficient information is sent in encrypted form so as to make it difficult to defeat. The encrypted messages include:

- SERVER-VERIFY
- CLIENT-CERTIFICATE
- REQUEST-CERTIFICATE

Depending on the situation, error messages can be encrypted or in cleartext, as described later in the chapter.

Once the session has been established, all further communications between the client and the server are encrypted.

134.9 Error Handling

Several errors can occur during the negotiations. These errors include:

- *NO-CIPHER-ERROR*. The client generates this error to the server indicating that there are no ciphers or key sizes supported by both ends of the connection. When this error occurs, the connection fails and cannot be recovered.

EXHIBIT 134.7 Domain name mismatch error.

EXHIBIT 134.8 SSL connection is not established.

- *NO-CERTIFICATE-ERROR*. When the server requests a certificate from the client and there is no certificate available, the client returns this error message to the server. The server can choose to continue with the connection, depending on the local configuration.
- *BAD-CERTIFICATE-ERROR*. This error is generated when the certificate cannot be verified by the receiving party due to a bad digital signature or inappropriate information in the certificate. A common example of bad information in the certificate is when the host name in the certificate does not match the expected name. This error can be recovered and is not uncommon. Exhibit 134.7 illustrates the results when a Web client cannot verify a server certificate. The user is presented with a window similar to this, where he must choose to accept the certificate or not. Should the user choose not to accept the certificate, a window similar to that shown in Exhibit 134.8 would be shown to the user. The connection between the client and the server is not established.
- *UNSUPPORTED-CERTIFICATE-TYPE-ERROR*. Occasionally a server or client may receive a certificate that it does not have support for. This error is returned to the originating system.

134.10 After the Handshake

Once the handshake is complete, the client and the server exchange their messages using the services of the SSL transport. Because SSL allows higher-level protocols to protect their data while in transport, SSL has been used for a variety of purposes, including protecting HTTP-based traffic and SSL VPN sessions.

134.10.1 SSL and the Web

The most well-known use of SSL is the protection of HTTP (World Wide Web) data when traveling across an untrusted network or carrying sensitive information. For example, E-commerce, secure online ordering, and bill payments are all performed on the Web using SSL as the protection layer.

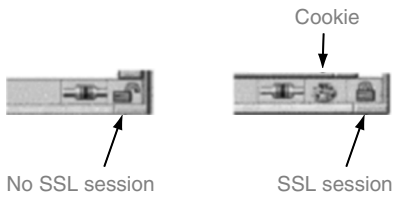

EXHIBIT 134.9 SSL on the Web.

The Web server must be capable of supporting SSL connections, and must have been properly configured with a server certificate, also known as a server-side certificate. The client specifies a Uniform Resource Locator (URL) with a https://prefix, indicating that the session is to be encapsulated within SSL.

The client contacts the Web server and the SSL handshake occurs. Once the SSL connection is established, the user sees a "key" or "lock" appear in the corner of their Web browser as seen in Exhibit 134.9.

Exhibit 134.9 illustrates a Web browser without an SSL connection, and the familiar lock indicating an SSL session has been established. Some Web servers will use SSL only for the specific transactions where protection is required, such as login forms, and credit card and E-commerce transactions.

134.10.2 SSL Tunnels

More recently, SSL has been used as the transport provider for virtual private networking. Commercial and open source software providers are including SSL VPN support in their products. One example is *stunnel*, an open source SSL VPN implementation for UNIX and Microsoft Windows-based systems.

SSL VPN solutions provide the same features as normal SSL applications, except the VPN implementation allows tunneling of non-SSL aware applications through the VPN to the target server or network. The VPN technology provides the encryption component, with no changes to the application required.

134.11 Attacking SSL

Like all network protocols and services, there are specific attacks that can be used against the SSL protocol or implementations of the protocol. Bear in mind that a weakness found in a specific implementation of the SSL protocol does not itself mean that SSL is flawed. What it means is that the implementation may be vulnerable to a specific attack or weakness, which does not inherently mean that all SSL implementations are vulnerable. For example, OpenSSL has been the subject of several attacks against its implementation of the protocol.

The attacks identified here do not constitute an all-inclusive list, but rather they represent some of the more commonly used attack methods that could be used to circumvent SSL.

134.11.1 Cipher Attacks

Because SSL uses several different technologies for the underlying encryption, attacks against the cryptographic engine or keys are inevitable. If a successful attack is found against any of the available cryptographic engines, SSL is no longer secure.

Consequently, any of the available methods of cryptographic analysis can be used. This includes recording a specific communications session and expending many CPU cycles to crack either the session or public key used.

Because many SSL sessions use 128-bit keys, the cost of launching an attack against a 128-bit key is still quite high. As new protocols and key lengths are supported within SSL, the work factor to defeat the cryptography increases.

134.11.2 Cleartext

Cleartext attacks are a fact of life with the SSL implementation. Because many messages in SSL are the same, such as HTTP GET commands, an attacker can build a dictionary where the entries are known values of specific words or phrases. The attacker then intercepts a session and compares the data in the session with the dictionary. Any match indicates the session key used and the entire data stream can be decrypted.

The work factor of the cleartext attack is quite high. For each bit added to the key, the dictionary size increases by a factor of two. This makes it virtually impossible to fabricate a dictionary with enough entries to defeat a 128-bit key using a cleartext attack methodology.

Given the high work factor associated with a cleartext attack, a brute-force attack, even with its high work factor, is considered the cheaper of the two. However, brute-force attacks also take an incredible amount of CPU horsepower and time. Even with today's high-speed computing equipment, the work factor associated with a brute-force attack against a 128-bit key is still considered an infinitely large problem.

134.11.3 Replay

Replay attacks involve the attacker recording a communication between the client and the server and later connecting to the server and playing back the recorded messages. While a replay attack is easy to originate, SSL uses a connection ID that is valid only for that connection. Consequently, the attacker cannot successfully use the recorded connection information. Because SSL uses a 128-bit value for the connection ID, an attacker would have to record at least 2^64 sessions to have a 50 percent chance of getting a valid session ID.

134.11.4 Man in the Middle

The man-in-the-middle attack (Exhibit 134.10) works by having the bad guy sit between the client and the server, with the attacker pretending to be the real server. By fooling the client into thinking it has connected to the real server, the attacker can decrypt the messages sent by the client, collect the data, and then retransmit it to the real server through an SSL session between the attacker and the real server.

The use of server certificates makes the man-in-the-middle attack more difficult. If the certificate is forged to match the real server's identity, the signature verification will fail. However, the attacker could create his or her own valid certificate, although it would not match the real server's name. If the certificate

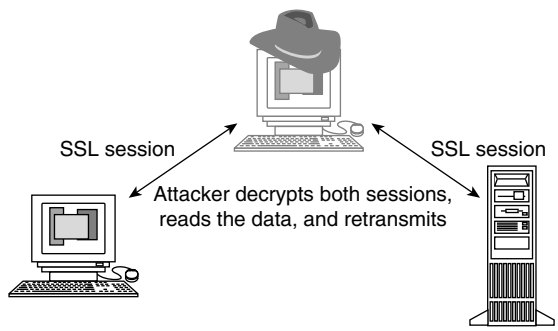

EXHIBIT 134.10 The man-in-the-middle attack.

matches the attacker but does not match the name, the user will see a window in his browser similar to Exhibit 134.7. If the user ignores the message, and many do, he will not be aware of the connection problem.

Consequently, organizations would do well to inform their users of the connection problems and issues associated with SSL and teach them to report problems when they are encountered. It is far better to report a configuration error than to realize later that the data was compromised.

134.12 The Cost of Encryption

Encryption of any form has a cost in performance—SSL included. If the SSL server experiences a high level of traffic, then the server itself may suffer performance degradation due to the load of performing the SSL encryption and decryption. This performance degradation can be addressed in a number of ways.

The first possibility is to redesign the application to limit the actual amount of data that is transferred via SSL. For example, a Web application may only require SSL on specific pages, and by switching SSL on and off when required, the server's performance can be increased. The danger in this approach is the possibility for data that should be protected to be missed. Only a thorough analysis of the application, data, and data flows can determine where the application must be SSL protected.

The second solution is to change the system or network architecture and implement SSL accelerator hardware to offload the primary CPU from the actual SSL operations. SSL accelerator hardware can be installed into the actual server hardware or implemented in the network to perform the SSL handshake and all the encryption/decryption operations. While this can be a more expensive approach, it does not require any re-design or thorough analysis of the application. Because SSL accelerators are often implemented in an application layer switch, other benefits can be achieved, including load balancing.

134.13 Policy

Any organization providing information to others on either a public or private network will need to consider the requirements for SSL. Many situations where it is necessary to encrypt data on the public network may apply to the private network as well. Consequently, organizations must consider their security policy and assist in determining when SSL is required.

For example, SSL should be used on the public network to protect every transaction containing any form of personal information about the user, financial data, or information that the organization does not want generally visible on the public network. Additionally, SSL should be used on the private network to protect employee data and any information potentially subject to privacy legislation.

Finally, any information exchange falling into the realm of HIPAA, Gramm–Leech–Bliley, or Sarbanes–Oxley within the United States should strongly consider the use of SSL due to its data integrity properties. However, the specific legislation for a country and an organization's data classification and security policies will assist in determining when and where SSL is required.

134.14 Summary

This chapter has presented how the Secure Socket Layer encryption facility works. Focused at the protocol level, the security professional should understand how SSL actually functions and the number of steps involved in achieving the SSL connection. SSL is used as the basis for protecting almost all encrypted Web traffic to prevent the loss of sensitive information in an untrusted network. It can easily be stated that Internet based E-commerce would not be where it is today without SSL.

SSL provides data confidentiality and integrity elements in the handshake to avoid successful attacks, although there is a certain degree of human intervention and understanding associated with doing the correct thing when problems occur. Additionally, once the SSL session is established, data is protected in the session from eavesdropping and it cannot be altered during transmit—alterations cause the decryption to fail at the receiving end, maintaining the integrity of the data.

Consequently, organizations should make use of SSL encryption whenever they work with data across an untrusted network such as the Internet and consider using it to protect sensitive data within their own network, as the same network threats apply.

Acknowledgments

The author thanks Mignona Cote, a trusted friend and colleague, for her support during the development of this chapter. Mignona continues to provide ideas and challenges in topic selection and application, always with an eye for practical application of the information gained. Her insight into system and application controls serves her and her team effectively on an ongoing basis.