Introduction to SSL

The cornerstone of e-commerce is a Web site's ability to prevent eavesdropping on data transmitted to and from its site. Without this, consumers would justifiably be afraid to enter credit card numbers, bank information, or other sensitive information on Web sites for fear that hackers could listen in on the transaction and steal sensitive information. SSL provides complex mathematical tools to secure transmissions from eavesdropping, which assures consumers and enables e-commerce to flourish.

SERICON technology

May 25, 2005

Overview

SSL

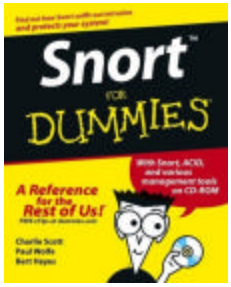
The standard method most e-commerce sites use to protect their transmitted data is the *Secure Sockets Layer* protocol (SSL), developed by Netscape in 1994.[1] Subsequently it was adopted by the Internet Engineering Task Force (IETF) as RFC 2246, commonly referred to as *Transport Layer Security* (TLS).[2]

Packet sniffing

When information is transmitted between computers, it is divided up into "packets" that travel separately through the Internet and are reunited at their destination. If you can get between the point of origin and the destination, you can use a *packet sniffer* to watch the traffic. If the data is unencrypted (i.e. if it does not use SSL), you can see the contents of these packets.

While this may sound rather difficult to do, it is in fact rather easy. There are even several packet sniffer programs available for free that can be used to do this. One popular program, Snort[3], even has a "For Dummies" book written about it.

Figure 1: Snort for Dummies book



We mention this book simply to show how mainstream this software is. In general, once a "For Dummies" book has been written about a computer program, that program can no longer be considered "exotic".

Protection against sniffing

SSL includes two methods for ensuring consumer confidence when performing e-commerce transactions: encryption and authentication.

- **Encryption:** The transmission of data should be secure so that no one can sniff the data that is sent. *Public Key Cryptography* (PKC) is a method that secures data transmission so that if data is sniffed it cannot be understood. PKC relies on an entity creating two keys that are used to encrypt information. These keys are related to each other in a very complicated mathematical way, but knowing one of the keys will not allow you to determine the other. One key is kept secret (the Private Key) while the other is made public (the Public Key). To send a secured message to this entity, you simply need to encrypt it using standard methods and that entity's public key. Once encrypted, the only way to decrypt the message is to use the private key. This means that the only one who can understand the message is the one with the private key. This is the basis for keeping SSL transmissions private. While encryption itself is a powerful tool, on its own, it is insufficient to give consumers the confidence they need when performing e-commerce transactions.

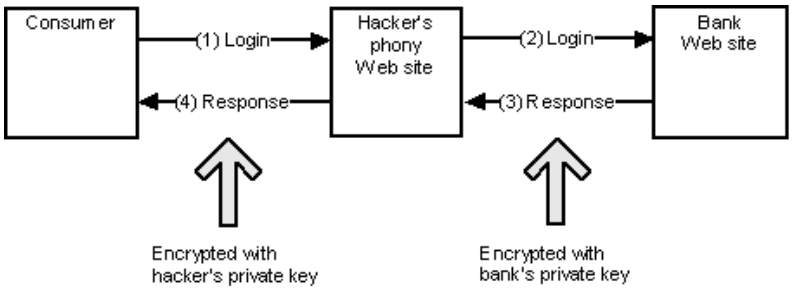
- **Authentication:** The user should be confident that data received was really sent by the correct Web site. This prevents a *man-in-the-middle* (MITM) attack, in which a hacker sits between a user and a legitimate Web server, while posing as the legitimate Web site. In reality, the user is connected to the

hacker's site which may be creating fictitious data, modifying the user's inputs, or even silently reading the traffic going between the user and the Web site.

To understand how a MITM attack works, it helps to first understand that on the Internet, any data passed between two computers is on a public network, and anyone with the desire and know-how can read it. A MITM attack occurs when a hacker positions himself between a victim and a resource that that victim wants to use. For instance, a consumer who wants to connect to a bank to conduct a transaction can browse to the bank's Web site and type in login account information. The hacker can then intercept this information and pass it on to the real bank Web site, thereby impersonating the real client. Whatever data is returned by the Web site is then forwarded (by the hacker) to the real consumer. In this way, the consumer is unaware that anything wrong is going on, and in fact, may even be communicating with the hacker in an encrypted manner. However, the hacker can see all transactions, and may be able to modify them for personal advantage.

Note that in this example, an encrypted link may exist between the hacker and the victim. This shows that encryption by itself is not a valid defense against packet sniffing, but rather authentication must also take place.

Figure 2: Data flow in a "man in the middle" attack



Of course, if this could really happen, no one would bank over the Internet. Internet banking is viable not only because encryption is used, but also because of authentication of the Web site with which there is an encrypted session. In other words, you can verify that the Web site is the one you want and not an imposter who has launched an MITM attack.

A Web site is generally authenticated by an X.509 certificate, as described below.

X.509 certificates

X.509 is a standard created by the International Telecommunication Union (ITU) and formalized by the IETF as RFC 2459.[4] An X.509 certificate contains information about the entity that owns it and binds it to that entity's public key. There is also data from a well-trusted third party confirming that all the information inside the certificate has been verified as true.

Web servers use certificates:

- to prove their identities to Web browsers
- to provide a public key to the browser so that it and the server may communicate securely

In this manner, X.509 certificates provide a mechanism on which an SSL session can be built. If an X.509 certificate contains the relevant data in order to create an SSL session, it can be considered to be an *SSL certificate*.

How certificates are created

To obtain an SSL certificate for a web server, the person running the server completes the following steps:

1. Generate a public and private key using a standard off-the-shelf tool, and put the public key into a certificate. Along with these keys, identifying information is entered in the certificate (the Subject information).
2. Send this certificate to an organization that is known to be trusted, called a *Certificate Authority* (CA). There are several companies that specialize in CA services, such as VeriSign Inc. of Mountain View, California and GeoTrust Inc. of Needham, Massachusetts.
3. The CA *vets* the certificate to positively confirm:
   - the identity of the sender through outside means (such as examining business documents)
   - the sender's authority to own this certificate
4. Only if the vetting process confirms the entity's identity, the CA *signs* the certificate and adds its identity to the Issuer field. By signing a certificate, the CA uses its private key to encrypt information into the certificate so that someone who examines it will be assured that that CA validated the certificate's information. Because the signing process requires possession of the CA's private key, which is highly guarded, it is not possible for someone to forge this signature.

Note: The CA can sign a certificate that is then used to sign additional certificates. This is known as a "chain."

Why must CAs sign certificates?

Since X.509 is an open standard, there are tools to create certificates containing any information you want. For example, it is relatively easy to create your own certificate that claims to belong to www.amazon.com and attempt to fool your friends with it. However, since a CA relies on public trust, it will not put its reputation on the line by signing a certificate unless it is sure of its validity. This means that if you create a certificate specifying Amazon.com as the owner and present it to a recognized CA for signing, it will be rejected unless you can prove that you have the authority to conduct business as Amazon.com.

Of course, this simply moves the problem up one level. In the same way that I can deviously create a phony Amazon.com certificate, I can create a phony VeriSign certificate and use it to sign my phony Amazon certificate. This would allow me to successfully launch a MITM attack. Fortunately, this deception is easy to detect.

To understand how to detect this scam, it is important to consider how it plays out. When you use your browser to access Amazon.com, and you are ready to pay for your purchase with a credit card, Amazon attempts to use SSL to secure the transaction: the Amazon server transmits its certificate to your browser, and if the certificate is trustworthy, the server switches to SSL mode and starts secured transmission. How does your browser know when a certificate is trustworthy? First it checks that the certificate is valid and is properly signed. If so, it checks that the Issuer is trustworthy. It can do that because each browser (e.g. Microsoft Internet Explorer, Firefox, etc.) has certificates from all known trustworthy CAs **built into the browser**. If the issuer is not from a certificate known to the browser, then a message box pops up warning the user that the certificate may not be valid. Thus, I can create a phony VeriSign certificate, but without VeriSign's private key (a closely guarded secret) I cannot fool any browser into thinking that my certificate is from the real VeriSign. This is the real protection against a MITM attack.

Figure 3: Security Alert from Microsoft Internet Explorer

How to view certificates

Each browser is shipped with a list of CA certificates pre-installed for immediate use. To see this list, you can go through the following browser-specific steps.

Microsoft Internet Explorer

1. Select **Tools | Internet Options**.
2. Click the **Content** tab.
3. Click the **Certificates** button.
4. Click the **Trusted Root Certification Authorities** tab.
5. Double-click any certificate in the list to see more information about it.

Mozilla Firefox

1. Select **Tools | Options**.
2. In the left pane, click the **Advanced** icon.
3. Expand **Certificates**.
4. Click the **Manage Certificates** button.
5. Click the **Authorities** tab.
6. Double-click any certificate in the list to see more information about it.

Data fields

Inside an X.509 certificate there are fields containing data that make up the certificate. To understand how a certificate works, we will examine several of these fields.

Subject

This field identifies the certificate owner. It contains information such as the name of the entity, the organization that it belongs to, and the geographical location in which it is located. For purposes of SSL, the Subject's Common Name is normally set to the domain name of the Web site that this certificate is associated with. For instance, if Amazon.com wanted to enable SSL transactions for its main Web server, it would have a certificate whose common name was `www.amazon.com`.

Note: Subject Common Name is not always the place in which a Web site's URL may be found. Other places may include the dNSName certificate extension.

Issuer

This field contains information about the entity that signed the certificate. It includes the same types of information as the Subject field.

Public key

This field contains the public key of the owner of this certificate. It is with this key that others may securely encrypt data and send it knowing that the owner is the only one who may successfully decrypt it.

Validity

This field contains the time period during which the certificate is valid.

Certificate revocation

Certificates are sometimes problematic. For example:

- The certificate was wrongly issued (because of information that was not known during the vetting process).
- The certificate has become compromised for some reason (such as someone stealing the private key).

In these cases, the certificate should be invalidated. If the end of the certificate's valid period is far in the future, this need becomes critical.

Revoking a certificate is difficult, since during an SSL session, the decision to start secure transmission is based on the data the server presents to the browser. The CA is not normally a party to this transaction, and it cannot change a certificate after signing it and transferring it back to its owner. The browser could check with the CA that each certificate it signs is valid each time it is accessed, although this defeats the purpose of embedding CA certificates into the browser and is very inefficient.

There are two popular methods of invalidating a signed certificate, which are described below.

CRL

The most common method for revoking a certificate is through a *Certificate Revocation List* (CRL). This is a list that a CA creates listing all the certificates it has signed that it now wants to revoke. This list is made available to all browsers, and can be automatically downloaded from the CA's Web site. In theory, browsers download and check a CA's CRL before trusting a certificate presented to it. In practice, this does not always happen properly. There are two problems with CRLs:

- Downloading a long list of revoked certificates from a Web site is a slow process. This ultimately slows the Web browsing experience.
- CAs only issue CRLs periodically. If a CA wants to revoke a certain certificate but the next CRL update is scheduled in another two weeks, this leaves a two-week window during which a bad certificate is still seen by browsers as trustworthy.

In reality, CRL usage is spotty at best, and many browsers do not properly implement this functionality due to the inherent problems. This creates vulnerability for all SSL users.

OCSP

Another way to revoke a certificate is through *Online Certification Status Protocol* (OCSP). This is a protocol in which the browser communicates with a server in real time to determine if a certificate is still trustworthy and not

revoked. Clearly, this process is very time consuming. In addition, it places a burden on the CA's servers. There is even less support for OCSP than for CRLs.

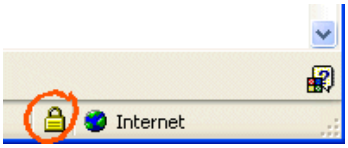
SSL in practice

As described above, SSL begins to work when a user connects to a server in which SSL is enabled. Before any secure data passes between them, the server sends its certificate to the browser for inspection. When the browser receives this certificate, it performs the following checks to verify its validity:

- The server certificate contains valid dates.
- The server certificate is properly signed by the issuer.
- The issuer is known to be trustworthy by the browser.
- No certificates in the chain have been revoked (by looking at either CRL or OCSP).
- The name on the server certificate exactly matches the hostname of the Web site to which the user is connected.

If all these checks are positive, then the browser can use the public key in the certificate to begin encrypted communication with the server. This means that SSL is being used, and the familiar padlock icon closes at the bottom of the browser window to inform the user that any communication is now secure.

Figure 4: Padlock icon in Internet Explorer indicating that SSL is in use



The padlock icon indicates:

- Encryption: transmission is encrypted so that no one can *sniff* it.
- Authentication: the browser is really connected to the server listed in the browser's address bar, and no hacker is pretending to be the server. (This is ascertained in the last check in the above list.)

This satisfies the initial requirements of SSL: By now, consumers are used to the padlock closing, and they have learned to demand this before typing any private information into their browsers, even if they do not explicitly understand what it means.

Failures

Sometimes when a browser inspects a server certificate, one of the checks fails. Common failures include:

- Invalid Dates

  If the current date is outside the range of the valid dates in the certificate, then this check fails. This does not necessarily indicate an imposter certificate: it may be caused by an incorrectly set clock on the user's computer. Alternatively, the Web site may have neglected to renew its certificate with the CA.
- Unknown Issuer

  If a browser encounters a CA certificate it does not recognize, then this check fails. This may indicate a dangerous situation, e.g. someone "posing" as a known Web site. Alternatively, the browser may be an old version, which does not have a known and trustworthy CA certificate built in, because it was issued after the browser was shipped.

When a failure occurs, the browser displays the relevant information to the user and enables the user to decide whether or not to proceed with the transaction. Unfortunately, most naïve users are ill-equipped to understand the reason for the failure. They are apt to go along with whatever is easiest without fully understanding the risks they may be taking. Therefore, any system that uses SSL should ensure that users are not burdened with the task of deciding what is valid or invalid.

Endnotes

[1] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professional, October, 2000.
[2] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. RFC 2246. January, 1999.
[3] http://www.snort.org
[4] R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. RFC 2459. January, 1999.