CSC 474 Network Security

Topic 8.3 SSL/TLS
Outline

I. Overview
II. The SSL Record Protocol
III. The SSL Handshake and Other Protocols
Overview of SSL
Reminder: What Layer?
Protocols

• Goal: application independent security
  – Originally for HTTP, but now used for many applications
  – Each application has an assigned TCP port, e.g., https (HTTP over SSL) uses port 443

• Secure Sockets Layer (SSL)
  – the de facto standard for web-based security
  – v3 was developed with public review

• Transport Layer Security (TLS)
  – TLS v1.0 very close to SSL v3.1
SSL Architecture

- Relies on TCP for reliable communication
Architecture (Cont’d)

- **Handshake protocol**: establishment of a session key
- **Change Cipher protocol**: start using the previously-negotiated encryption / message authentication
- **Alert protocol**: notification (warnings or fatal exceptions)
- **Record protocol**: protected (encrypted, authenticated) communication between client and server
SSL Services

- Peer authentication
- Negotiation of security parameters
- Generation / distribution of session keys
- Data confidentiality
- Data integrity
Connections and Sessions

- **SSL Session**
  - an association between peers
  - created through a handshake, negotiates security parameters, can be long-lasting
- **SSL Connection**
  - a type of service (i.e., an application) between a client and a server
  - transient
- Multiple connections can be part of a single session
Session Parameters

- Session ID
- X.509 public-key certificate of peer
- Compression algorithm to use
- Cipher specification: encryption algorithm, message digest, etc.
- Master (session) secret: 48-byte (384 bits) secret negotiated between peers
Connection Parameters

- Server and client *nonces*
- Server and client *authentication keys*
- Server and client *encryption keys*
- Server and client *initialization vectors*
- Current message *sequence number*
Ciphers Supported by SSL

- DES+HMAC/SHA-1
- 3DES+HMAC/SHA-1
- RC4+MD5
- RC2+MD5
- +others
The SSL Record Protocol
Protocol Steps

1. Fragment data stream into records
   – each with a maximum length of $2^{14}$ (=16K) bytes
2. Compress each record
3. Create message authentication code for each record
4. Encrypt each record
Steps… (cont’d)

Application Data

Fragment

Compress

Add MAC

Encrypt

Add SSL Hdr
SSL Record Format

- There is, unfortunately, some version number silliness between v2 and v3; see text for (ugly) details

<table>
<thead>
<tr>
<th>Record Type</th>
<th>SSL Version</th>
<th>Payload Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Application Data (optionally compressed)
- Optional MAC (16 or 20 bytes)
Possible Record “Payloads”

(a) Change Cipher Spec Protocol

(b) Alert Protocol

(c) Handshake Protocol

(d) Other Upper-Layer Protocol (e.g., HTTP)
SSL Handshake Protocol
Phases of Protocol

I. Establish security capabilities
   • version of SSL to use
   • cipher + parameters to use

II. Authenticate server (optional), and perform key exchange

III. Authenticate client (optional), and perform key exchange

IV. Finish up
All the Messages

Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.
I. Establish Security Capabilities

- Messages marked with * are mandatory
Client_Hello Message

• Transmitted in plaintext
• Contents
  – highest SSL version understood by client
  – $R_C$: a 4-byte timestamp + 28-byte random number
  – session ID: 0 for a new session, non-zero for a previous session
  – list of supported cryptographic algorithms
  – list of supported compression methods
Server Hello Message

- Also transmitted in plaintext
- Contents
  - minimum of (highest version supported by server, highest version supported by client)
  - $R_S$: 4-byte timestamp and 28-byte random number
  - session ID
  - a cryptographic choice selected from the client’s list
  - a compression method selected from the client’s list
II. Server Auth. / Key Exchange

- The Server_Certificate message is optional, but almost always used in practice
Server Certificate Message

- Contains a certificate with server’s public key, in X.509 format
  - or, a chain of certificates if required
- The server certificate is necessary for any key exchange method except for anonymous Diffie-Hellman
Authenticating the Server

- Step #4: Domain name in certificate **must** match domain name of server (not part of SSL protocol, but clients should check this)
Key Exchange Methods Supported

- **RSA** (server must have a certificate)
- **Ephemeral Public Key**
  - public keys are exchanged, signed using long-term RSA keys
- **(Fixed Diffie-Hellman)**
  - server provides the D-H public parameters in a certificate
  - client responds with D-H public key either in a certificate, or in a key exchange message
- **Anonymous Diffie-Hellman**
Server_Key_Exchange Message

• Needed for…
  – anonymous D-H
  – ephemeral public key
# Server Key Exchange

<table>
<thead>
<tr>
<th>Handshake</th>
<th>Server Key Exchange (Diffie-Hellman)</th>
<th>Server Key Exchange (RSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>p (modulus, prime)</td>
<td>m (modulus = p*q)</td>
</tr>
<tr>
<td>Length</td>
<td>g (generator)</td>
<td>e (pub. exp.)</td>
</tr>
<tr>
<td>Data</td>
<td>g^{as} mod p</td>
<td>Signature</td>
</tr>
<tr>
<td></td>
<td>Signature</td>
<td></td>
</tr>
</tbody>
</table>

## Diffie-Hellman

- Client Computes: \( \text{PreMasterSecret} = (g^{as})^{ac} \mod p \)
- Client Sends: \( g^{ac} \) to server
- Server Computes: \( \text{PreMasterSecret} = (g^{ac})^{as} \mod p \)

## RSA

- Client Computes: \( y = \text{PreMasterSecret}^e \mod p \)
- Client sends: \( y \) to server
- Server Computes: \( \text{PreMasterSecret} = y^d \mod p \)
Client_Certificate_Request Msg.

• Normally not used, because in most applications
  – only the server is authenticated
  – client is authenticated at the application layer, if needed

• Two parameters
  – certificate type accepted, e.g., RSA/signature only, DSS/signature only, …
  – list of certificate authorities recognized (i.e., trusted third parties)
III. Client Auth. / Key Exchange

Client

Server

Client Certificate

Client Key Exchange*

Client Certificate Verify
Client Certificate Message

• Contains a certificate, or chain of certificates if needed
Client_Key_Exchange Message

• If using RSA, the pre-master secret S, encrypted with the server’s public key
• If using D-H, the client’s public key
Client_Certificate_Verify Msg

- Proves the client is the valid owner of a certificate (i.e., knows the corresponding private key)
- Only sent following any client certificate that has signing capability
IV. Finish Up

Switch to the negotiated cipher for all remaining (application) messages
**Change_Cipher_Spec** *Msg*

- Confirms the change of the current state of the session to a newly-negotiated set of cryptographic parameters

- **Finished** Messages
  - keyed hash of the previous handshake messages to prevent man-in-the-middle-attacks from succeeding
“Abbreviated” Protocol Possible

- Allows **resumption** of a previously-established session
  - does not require authentication of server or client
  - does not exchange keys
- Details omitted
Creating the “Master” Secret

• The master secret is a one-time (per session) 48-byte (= 16+16+16) value

• Parameters
  – the pre-master secret S has previously been communicated using RSA or D-H
  – the client nonce R\textsubscript{c}
  – the server nonce R\textsubscript{s}

• Computation: K =
  \[
  \text{MD5} (S \ | \ \text{SHA-1}(“A” \ | \ S \ | \ R\textsubscript{c} \ | \ R\textsubscript{s})) \ | \\
  \text{MD5} (S \ | \ \text{SHA-1}(“BB” \ | \ S \ | \ R\textsubscript{c} \ | \ R\textsubscript{s})) \ | \\
  \text{MD5} (S \ | \ \text{SHA-1}(“CCC” \ | \ S \ | \ R\textsubscript{c} \ | \ R\textsubscript{s}))
  \]
Cryptographic Parameters

- Generated from
  - the master secret K
  - Rc
  - Rs

- Values to be generated
  - client authentication and encryption keys
  - server authentication and encryption keys
  - client encryption IV
  - server encryption IV
Alert Protocol Examples

• Type 1: **Fatal_ALERT**
  – ex.: *Unexpected_Message, Bad_MAC*, etc.
  – connection is immediately terminated

• Type 2: **Warning**
  – ex.: *No_Certificate, Close_Notify*
Summary

1. SSL is the de facto authentication/encryption protocol standard for HTTP
   – becoming popular for many other protocols as well

2. Allows negotiation of cryptographic methods and parameters